Tivoli® Inventory User's Guide

Version 4.0

Tivoli Inventory User's Guide

Copyright Notice

© Copyright IBM Corporation 2001. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement, an IBM Software License Agreement, or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of IBM Corporation. IBM Corporation grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the IBM Corporation copyright notice. No other rights under copyright are granted without prior written permission of IBM Corporation. The document is not intended for production and is furnished "as is" without warranty of any kind. **All warranties on this document are hereby disclaimed, including the warranties of merchantability and fitness for a particular purpose.**

U.S. Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corporation.

Trademarks

IBM, the IBM logo, Tivoli, the Tivoli logo, AIX, OS/2, Tivoli Enterprise, Tivoli Enterprise Console, and TME are trademarks or registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel is a trademark of Intel Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Notices

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to valid intellectual property or other legally protectable right of Tivoli Systems or IBM, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user. Tivoli Systems or IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.

Contents

Chapter 1—Introduction to Tivoli Inventory

Tivoli Inventory and Scalable Collection Service Components 1-2
- Tivoli Inventory Components 1-2
- SCS Components 1-3

How Does Tivoli Inventory Work? 1-5
- Using Tivoli Inventory with Scalable Collection Service 1-6

Tivoli Inventory and SCS Features 1-10
- Tivoli Inventory Features 1-10
- SCS Features 1-13

Tivoli Inventory and Other Tivoli Applications 1-14

Security 1-14

Chapter 2—Installing Tivoli Inventory, Version 4.0

Overview 2-1

Coexistence with Previous Product Versions 2-2

Using Tivoli Software Installation Service 2-3

Installing Scalable Collection Service 2-4

Installing the SCS Patch 2-6
- Selecting the Media 2-6
- Installing Tivoli Scalable Collection Service For Framework 3.7.1 2-8

Using RIM and the RDBMS 2-10
- Choosing a RIM Host 2-11
- Choosing an RDBMS Server 2-11

Configuring an RDBMS 2-11
- Configuring DB2 2-12
- Configuring Informix 2-14
- Configuring MS SQL Server 2-15
- Configuring Oracle 2-15
- Configuring Sybase 2-16

Preparing to Install Tivoli Inventory Products on Target Systems 2-17
Selecting the Media 2-18
Choosing Between Tivoli Inventory Server and Tivoli Inventory Gateway 2-21
Installing Tivoli Inventory Server on the TMR Server and Managed Nodes ... 2-21
Desktop 2-22
Changing Passwords for RIM Objects 2-28
Command Line 2-29
Installing Tivoli Inventory Gateway 2-31
Removing Tivoli Inventory 2-33
Removing Tivoli Inventory with an Automated Uninstall 2-33
Removing Systems from the Configuration Repository 2-35
Specifying Tivoli Enterprise Console Servers 2-35
Where To Go From Here 2-37

Chapter 3—Creating the Configuration Repository

Creating a DB2 Configuration Repository 3-2
Creating the User invtiv and the Tivoli Inventory Database 3-3
Installing the Configuration Repository Schema 3-4
Testing the Configuration 3-6
Where to Go from Here 3-7
Creating an Informix Configuration Repository 3-7
Creating the Informix User and the Tivoli Inventory Database 3-8
Installing the Configuration Repository Schema 3-8
Testing the Configuration 3-9
Where to Go from Here 3-9
Creating an MS SQL Server Configuration Repository 3-9
Creating the User invtiv and the Tivoli Inventory Database 3-10
Installing the Configuration Repository Schema 3-11
Testing the Configuration 3-12
Where to Go from Here 3-13

Creating an Oracle Configuration Repository 3-13
Creating the User invtiv and the Tivoli Inventory Database 3-13
Installing the Configuration Repository Schema 3-15
Testing the Configuration 3-16
Where to Go from Here 3-17
Creating a Sybase Configuration Repository 3-17
Creating the User invtiv and the Tivoli Inventory Database 3-17
Installing the Configuration Repository Schema 3-19
Testing the Configuration 3-20
Where to Go from Here 3-21
Installing Software Signatures 3-21
Installing Tivoli Inventory and Subscription Queries 3-22
Where to Go from Here 3-25

Chapter 4—Working with Scalable Collection Service

How SCS Works with Tivoli Inventory 4-2
Using Collectors 4-4
Stopping, Starting, and Resetting a Collector 4-4
Configuring a Collector 4-5
Using the Inventory Data Handler 4-7
Configuring the Inventory Data Handler 4-8
Moving the Inventory Data Handler 4-10
Moving the Inventory Callback Object 4-10
Creating and Configuring RIM Objects for the Inventory Data Handler 4-11
Creating RIM Objects 4-12
Configuring RIM Objects 4-13
Scheduling Collections 4-14
Controlling the Flow of Network Data 4-17
Obtaining Information about SCS 4-18
Collector Configuration 4-19
Collector Status 4-19
CTOCs and Queues 4-19

Inventory Data Handler Configuration 4-20
Inventory Scans 4-20
Inventory Profiles 4-21
Data Stored in the Configuration Repository 4-21
Configuring Tivoli Inventory Status Information 4-21
Considerations for Interconnected TMRs 4-22
Where to Go from Here 4-23

Chapter 5—Working with Inventory Profiles

Using Inventory Profiles 5-2
Creating an Inventory Profile 5-3
Desktop 5-4
Command Line 5-7
Customizing an Inventory Profile 5-8
Customizing Global Properties 5-9
Customizing a Software Scan 5-10
Customizing a Hardware Scan 5-18
Using Custom Scripts and MIF Files 5-23
Authorization Roles 5-24
Desktop 5-25
Command Line 5-25
Cloning an Inventory Profile 5-25
Desktop 5-26
Command Line 5-26
Deleting an Inventory Profile 5-27
Desktop 5-28
Command Line 5-29
Renaming an Inventory Profile 5-29
Where to Go from Here 5-29

Chapter 6—Distributing Inventory Profiles

Distributing Inventory Profiles 6-1
Desktop 6-3
Command Line 6-7
Performing an Endpoint-Initiated Scan 6-8
Populating the Configuration Repository 6-9
Where to Go from Here 6-10

Chapter 7—Collecting Custom Information with Tivoli Inventory

Working with Software Signatures 7-2
Desktop 7-3
Command Line 7-4
Gathering User Information 7-5
Overview 7-6
Planning to Collect User Information 7-7
Giving Users Access to UserLink 7-8
Designing Custom Tables 7-14
Naming Custom Tables, Views, and Columns for DB2 and Informix 7-16
Creating History Tables for Custom Tables 7-19
Adding a Custom Table to the Configuration Repository 7-20
Creating MIF Groups and Attributes with the User Data Template 7-22
Using the userlink.htm File 7-31
Completing the User Data Form 7-31
Retrieving and Saving User Information 7-34
Viewing User Information 7-34
Using Custom MIF Files 7-36
Customizing PC Scans for BIOS Information 7-38
Where to Go from Here 7-39

Chapter 8—Querying Inventory Information

Using the Inventory and Subscription Queries 8-2
Creating a Query 8-2
Desktop 8-4
Command Line 8-10
Running a Query 8-10
Desktop 8-11
Command Line 8-13
Defining Subscribers with Queries 8-14
Desktop 8-14
Command Line 8-16
Querying for Information about One Client 8-16
Desktop 8-18
Command Line 8-19
Using Queries in Interconnected TMRs 8-20

Appendix A—Authorization Roles

Inventory Roles A-1
Inventory Operations A-2
Query Operations A-3

Appendix B—Commands

Using Tivoli Commands B-1
Command Line Syntax B-2
Example B-2
Object References B-2
Registered Names B-3
Object Paths B-4
Examples B-4
Tivoli Inventory Commands B-4
SCS-Related Commands B-6

Command Syntax B-7
wcollect B-8
wcrtinvcb B-16
wcrtinvdh B-18
wcstat B-22
wdistinv B-25
wepscan B-31
wgetinvdh B-34
wgetinvglobal B-37
wgetinvpcfiles B-41
wgetinvpchw B-43
wgetinvpcsw B-46
wgetinvunixfiles B-51
wgetinvunixhw B-53
wgetinvunixsw B-56
wgetscanstat B-61
winvfilter B-63
winvrmnode B-65
winvsig B-67
wqueryinv B-71
wsetinvdh B-73
wsetinvglobal B-76
wsetinvpcfiles B-81
wsetinvpchw B-86
wsetinvpcsw B-90
wsetinvunixfiles B-95
wsetinvunixhw B-100
wsetinvunixsw B-104

Appendix C—Configuration Repository

History Tracking C-2
- Using History Tables C-3
- Custom History Tables C-4
- Deleting History Tables C-4

Configuration Repository Views C-5
- CDROM_VIEW C-6
- COMPUTER_MEM_VIEW C-7
- COMPUTER_VIEW C-9
- FLPY_DRV_VIEW C-10
- HDISK_VIEW C-11
- HEADER_INFO_VIEW C-13
- INST_FILE_VIEW C-14
- INST_SWARE_VIEW C-16
- INVENTORYDATA C-17
- IP_ADDR_VIEW C-19
- IPX_ADDR_VIEW C-21
- KEYBOARD_VIEW C-22
- MATCH_SWARE_VIEW C-23
- MOUSE_VIEW C-24
- NATIV_SWARE_VIEW C-25
- NET_CARD_VIEW C-26
- NOSIG_FILES_VIEW C-28
- NW_SERVER_VIEW C-29
- NW_VOLS_VIEW C-32
- OS_VIEW C-34
- PARTITION_VIEW C-36
- PC_BIOS_VIEW C-38
- PC_PROCESSOR_VIEW C-40
- PCI_DEV_VIEW C-44
- PRINTER_VIEW C-45

PROCESSOR_VIEW....C-47
STORAGE_DEV_VIEW....C-48
SWARE_MATCH_CRC32....C-49
SWARE_MATCH_MD5....C-51
SWARE_MATCH_QUICK....C-53
USB_DEV_VIEW....C-55
VID_CARD_VIEW....C-57
Historical Configuration Repository Views....C-58
Tivoli Inventory and Subscription Queries....C-60
Tivoli Inventory Queries....C-60
Subscription Queries....C-80
Configuration Repository Tables....C-83
COMPUTER....C-83
COMPUTER_SYS_MEM....C-84
FILE_DESC....C-85
FILE_FILTER....C-85
FILE_PATH....C-86
HDISK....C-86
HEADER_INFO....C-87
INST_HEADER_INFO....C-87
INST_MOUSE....C-88
INST_NATIV_SWARE....C-88
INST_PARTITION....C-89
INST_PRINTER....C-89
INST_PROCESSOR....C-90
INST_USB_DEV....C-90
INST_VID_CARD....C-91
IP_ADDR....C-92
IPX_ADDR....C-92
MATCHED_SWARE....C-93
MOUSE....C-93

NATIV_SWARE C-94
NET_ADAPTER C-94
NW_SERVER C-95
NW_VOLS C-96
PC_SYS_PARAMS C-96
PCI_DEV C-97
PRINTER C-98
PROCESSOR C-98
STORAGE_DEV C-100
SWARE_SIG C-100
UNIX_SYS_PARAMS C-101
UNMATCHED_FILES C-102
USB_DEV C-103
VID_CARD C-103
History Tables C-104

Appendix D—Policy

Default Policy Methods D-2
Policy Objects D-2
Creating a New Policy Object D-3
Replacing the Contents of a Policy Method D-4
Assigning Policy to a Policy Region D-5
Example—Setting a Default Policy Method D-6
Policy Methods D-11
ic_def_global_action D-13
ic_def_global_ep_timeout D-14
ic_def_global_logfile_host D-15
ic_def_global_logfile_path D-16
ic_def_global_notice_interval D-17
ic_def_global_notice_location D-18
ic_def_global_notice_type D-19
ic_def_global_update_replace D-20

ic_def_pc_custom_mifs D-21
ic_def_pc_custom_script D-22
ic_def_pc_exclude_dirs D-23
ic_def_pc_extensions D-24
ic_def_pc_hw_gran D-25
ic_def_pc_hw_outfile D-26
ic_def_pc_hw_scan D-27
ic_def_pc_include_dirs D-28
ic_def_pc_sw_crc D-29
ic_def_pc_sw_flags D-30
ic_def_pc_sw_outfile D-31
ic_def_pc_sw_scan D-32
ic_def_unix_custom_mifs D-33
ic_def_unix_custom_script D-34
ic_def_unix_exclude_dirs D-35
ic_def_unix_extensions D-36
ic_def_unix_hw_gran D-37
ic_def_unix_hw_outfile D-38
ic_def_unix_hw_scan D-39
ic_def_unix_include_dirs D-40
ic_def_unix_sw_crc D-41
ic_def_unix_sw_flags D-42
ic_def_unix_sw_outfile D-43
ic_def_unix_sw_scan D-44

Appendix E—Troubleshooting

Configuration Repository Schema E-1
DMI E-1
    Adding a DMI Layer E-1
    DMI Scans E-3
    DMI Table and Column Names E-4

Endpoint-Initiated Scans E-4
Graphical User Interface E-6
System Requirements E-6
Enabling the Log File E-6
Enabling the GUI E-7
Scalable Collection Service E-8
Log and Data Files E-8
Depot Contents E-13
Troubleshooting Procedures E-13
Queries E-19

Glossary

Index

Preface

The *Tivoli Inventory User's Guide* provides information about how to manage hardware and software inventory on PCs and UNIX systems. Tivoli Inventory uses profiles to scan for information and the Tivoli Inventory configuration repository to store information. This guide includes an introduction to the product, installation instructions, and information about, procedures for, and examples of the management tasks that you can complete using Tivoli Inventory.

Who Should Read This Guide

The target audiences for this guide are system administrators and other users of Tivoli Inventory who perform inventory gathering and reporting operations about PC and UNIX systems in a distributed enterprise. You should have some knowledge of the following:

- UNIX and PC architectures
- Basic inventory control and system configuration management concepts
- Database and SQL concepts
- Graphical user interfaces

Prerequisite and Related Documents

Before beginning the Tivoli Inventory installation, read the *Tivoli Inventory Release Notes, Version 4.0*. The release notes include the following information:

- New features in the Tivoli Inventory, Version 4.0, release
- System requirements, including supported platforms, Web browsers, and configuration repository space requirements
- Installation notes
- Defects fixed by Tivoli Inventory 4.0
- Known defects and workarounds
- Product notes about Tivoli Inventory 4.0

Also, refer to the following prerequisite and related documentation for more information about the management options that Tivoli Inventory provides:

- *Tivoli Management Framework Planning for Deployment Guide*, *Tivoli Enterprise Installation Guide*, *Tivoli Management Framework User's Guide*, *Tivoli Management Framework Reference Manual*, and *Tivoli Management Framework Maintenance and Troubleshooting Guide*.

  Includes prerequisite information about setting up and using the Tivoli environment.

- *Tivoli Software Distribution User's Guide*.

  Provides related information about using Tivoli to deploy software.

- *Tivoli Inventory Online Help*.

  Provides related information about using the Tivoli Inventory graphical user interface (GUI).

What This Guide Contains

The *Tivoli Inventory User's Guide* contains the following sections:

- Chapter 1, "Introduction to Tivoli Inventory"

  This chapter provides an overview of how Tivoli Inventory works, a list of features, a description of how Tivoli Inventory works with other Tivoli applications, and information about configuring security for Tivoli Inventory tasks and information.

- Chapter 2, "Installing Tivoli Inventory, Version 4.0"

  This chapter provides the steps necessary to install Tivoli Inventory, Version 4.0, in your Tivoli environment.

- Chapter 3, "Creating the Configuration Repository"

  This chapter provides the steps necessary to create the Tivoli Inventory configuration repository schema, install software signatures in the configuration repository, and install the queries that you can use to obtain information from the configuration repository.

- Chapter 4, “Working with Scalable Collection Service”

  This chapter provides information about configuring and using Scalable Collection Service (SCS), a Tivoli Management Framework service that enables efficient, asynchronous collection of large amounts of data across complex networks.

- Chapter 5, “Working with Inventory Profiles”

  This chapter describes the steps needed to create, customize, clone, and delete inventory profiles.

- Chapter 6, “Distributing Inventory Profiles”

  This chapter provides the steps that you take to distribute inventory profiles from the Tivoli desktop and from UserLink. It also describes how to run an endpoint-initiated scan.

- Chapter 7, “Collecting Custom Information with Tivoli Inventory”

  This chapter provides the procedures for collecting custom and optional information with Tivoli Inventory. You can collect custom information by adding software signatures to the configuration repository. You can also add custom tables, collect custom information with UserLink, and populate the custom tables. Finally, this chapter also describes how users can provide custom information to you.

- Chapter 8, “Querying Inventory Information”

  This chapter describes the tasks necessary to search for information in the configuration repository. The Tivoli Management Framework query facility enables you to construct SQL queries to retrieve information gathered from inventory profile distributions.

- Appendix A, “Authorization Roles”

  This appendix contains tables that describe the roles you need to perform activities using Tivoli Inventory.

- Appendix B, “Commands”

  This appendix lists and documents, in alphabetical order, the Tivoli commands related to Tivoli Inventory and SCS.

- Appendix C, “Configuration Repository”

  This appendix lists and describes each table and view included in the configuration repository schema for Tivoli Inventory, Version 4.0. This appendix also lists and describes the queries that you can install when you install Tivoli Inventory, Version 4.0.

- Appendix D, “Policy”

  This appendix includes an explanation of policies along with a description of each policy included with Tivoli Inventory.

- Appendix E, “Troubleshooting”

  This appendix provides troubleshooting information for Tivoli Inventory.

- Glossary

  The glossary contains terms and definitions relevant to Tivoli Inventory.

Conventions Used in This Guide

The guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

Bold

Commands, keywords, authorization roles, or other information that you must use literally appear in **bold**.

Italics

Variables, values that you must provide, and new terms appear in *italics*. Words and phrases that are emphasized also appear in *italics*.

`Monospace`

Code examples, output, and system messages appear in a `monospace` font.

Many procedures include icons in the left margin to provide a visual hint for performing a step within a procedure. For example, a policy region icon appears in the left margin next to the first step in a procedure that requires you to double-click on a policy region icon.

Accessing Publications Online

The Tivoli Customer Support Web site (http://www.tivoli.com/support/) offers a guide to support services (the *Customer Support Handbook*); frequently asked questions (FAQs); and technical information, including release notes, user's guides, redbooks, and white papers. You can access Tivoli publications online at http://www.tivoli.com/support/documents/. The documentation for some products is available in PDF and HTML formats. Translated documents are also available for some products.

To access most of the documentation, you need an ID and a password. To obtain an ID for use on the support Web site, go to http://www.tivoli.com/support/getting/.

Resellers should refer to http://www.tivoli.com/support/smb/index.html for more information about obtaining Tivoli technical documentation and support.

Business Partners should refer to "Ordering Publications" for more information about obtaining Tivoli technical documentation.

Ordering Publications

Order Tivoli publications online at http://www.tivoli.com/support/Prodman/html/pub_order.html or by calling one of the following telephone numbers:

- U.S. customers: (800) 879-2755
- Canadian customers: (800) 426-4968

Providing Feedback about Publications

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send e-mail to pubs@tivoli.com.
- Fill out our customer feedback survey at http://www.tivoli.com/support/survey/.

Contacting Customer Support

The *Tivoli Customer Support Handbook* at http://www.tivoli.com/support/handbook/ provides information about all aspects of Tivoli Customer Support, including the following:

- Registration and eligibility.
- How to contact support, depending on the severity of your problem.
- Telephone numbers and e-mail addresses, depending on the country you are in.
- What information you should gather before contacting support.

1

Introduction to Tivoli Inventory

The complexity of client/server management increases as enterprise-wide networks span broad geographic locations and incorporate systems and software from a variety of vendors. Tivoli Management Framework is a distributed, object-oriented foundation designed specifically to deploy secure, heterogeneous, and scalable systems management applications. These applications, such as Tivoli Inventory, provide solutions for managing a complex client/server environment. Tivoli Inventory also uses Scalable Collection Service (SCS), formerly referred to as MCollect. SCS is a Tivoli Management Framework service that enables efficient, asynchronous collection of large amounts of data across complex networks

This chapter provides an overview of how Tivoli Inventory and SCS work, a list of features, a description of how Tivoli Inventory and SCS work with other Tivoli applications, and information about configuring security for Tivoli Inventory tasks and information.

Using traditional methods to assess inventory, you must travel to each system, write down the software and hardware inventory information you collect, and enter the information into a spreadsheet or database program. When users upgrade software and hardware, you must update the spreadsheet or database.

This method is time-consuming and difficult to keep current. As a result, administrators and accounting personnel cannot automatically reuse inventory information to perform system and software upgrades and management tasks. However, with Tivoli Inventory, gathering and

maintaining up-to-date inventory information in a distributed environment is quick, accurate, and easy.

Tivoli Inventory gathers hardware and software inventory to help system administrators and accounting personnel manage complex distributed client/server enterprises. With Tivoli Inventory, administrators and accounting personnel can perform the following tasks:

- Manage all enterprise systems from a central point
- Determine the installed software base
- Confirm a software distribution
- Supplement and replace physical inventory functions
- Assist in procurement planning
- Check software requirements
- Control assets

Tivoli Inventory and Scalable Collection Service Components

The following sections describe the components you need to use Tivoli Inventory and to use SCS for Tivoli Inventory collections.

Tivoli Inventory Components

Tivoli Inventory relies on the following components to collect and store information:

- The Tivoli Management Region (TMR) server, which is responsible for issuing scan requests to the target systems.
- The Tivoli Inventory application, which you must install on at least the TMR server. The application includes both the graphical user interface (GUI) components and the command line interface (CLI) utilities.
- The Tivoli Inventory Gateway application, which you must install on any managed node that you want to act as a gateway for distributing inventory profiles to endpoints.

- The relational database management system (RDBMS) server and the Tivoli Inventory *configuration repository* in the RDBMS. The configuration repository is the RDBMS that contains the schema (tables and columns) in which software and hardware inventory information is stored. The configuration repository schema provides a structure for storing information collected during an inventory scan.
- The *RDBMS Interface Module (RIM) host*, which enables Tivoli Inventory to communicate with an RDBMS.
- *UserLink for Tivoli Inventory*, which enables users to perform some Tivoli Inventory-related tasks including completing the User Data Form.
- The Web-based interface for administrators, which enables administrators to perform some Tivoli Inventory-related tasks from the Web. Administrators can create a User Data Template to collect custom information from users through UserLink for Tivoli Inventory.

SCS Components

Any environment that uses SCS for Tivoli Inventory collections must include the following components:

- *Repeater sites* organized into a *repeater hierarchy*—systems that use the *multiplexed distribution* (MDist 2) service. MDist 2 parameters control the way that information is distributed throughout a Tivoli environment. A repeater hierarchy is the order in which information flows from one repeater to the next and then to the endpoints that are targets of the distributed data.

  SCS uses a *collector hierarchy* that mirrors the MDist 2 repeater hierarchy. SCS sends data upstream through this hierarchy, in the opposite direction of MDist 2 distributions.

  See the *Tivoli Management Framework Planning for Deployment Guide* for more information about configuring a repeater hierarchy.
- *Collectors*—repeater sites on which SCS has been installed. Specifically, a collector is an SCS daemon process on either a

managed node or gateway that stores and then forwards data to other collectors or to the inventory data handler.

This guide refers to any managed node as a collector if SCS has been installed on the node and the node has been added to the SCS collection hierarchy.

Collectors are composed of the following components:

- The *depot*, which persistently stores data that has been collected from endpoints or other collectors. The depot also sends data to other collectors that request it.
- The *queues*, which hold the *collection table of contents* (CTOCs). CTOCs contain, among other things, the data file name and size and the source and destination of the collected data. The input queue controls the order in which CTOCs are processed for collection. The output queue controls the order of the CTOCs as they are sent out from the collector. The completed, deferred, and error queues hold CTOCs for completed and deferred data collection and error conditions, respectively.
- A multithreaded scheduler daemon that processes input and output queues and controls data flow through the collector depot.

■ A *collection manager*, which maintains the collector hierarchy based on repeater hierarchy information obtained from the MDist 2 repeater manager.

■ An *inventory data handler*—the Tivoli Inventory object that receives data from collectors and sends the data to one or more RIM objects. (In Tivoli Inventory, Version 3.6.2, this object was called the inventory receiver.) The inventory data handler can be considered the final collector in a Tivoli Inventory system. Like collectors, the inventory data handler has a depot and queues. However, the inventory data handler uncompresses and decodes the data and sends it to one or more RIM objects rather than requesting collection from an upstream collector.

■ The *status collector*, which collects, stores, and distributes status information for each endpoint in a scan. You can configure the

status collector to keep lists of completed scans, successful scans, failed scans, and error messages. The status collector maintains this information throughout the scan, so scan status information is available during the scan.

The status collector is installed on the same managed node as the inventory data handler. If that node fails and is restarted, status information tracked by the status collector is automatically restored.

- The *inventory callback object*, which performs the following functions:
  - When inventory data cannot be collected using the collector hierarchy, MDist 2 sends the data to the inventory callback object, which sends it to the inventory data handler. The data is then sent through one or more RIM objects to the configuration repository.
  - For all scans, MDist 2 sends a status message to the inventory callback object indicating whether it successfully delivered the inventory profile to the endpoint. This message indicates only that the endpoint processed the profile, not that the scan data successfully reached the configuration repository.
- One or more RIM objects, which connect Tivoli Inventory to the RDBMS for access to the Tivoli Inventory configuration repository. You can configure multiple RIM objects to write SCS data in parallel to the configuration repository.

How Does Tivoli Inventory Work?

Tivoli Inventory relies on resources and management concepts provided by the Tivoli Management Framework. Tivoli Inventory supports the concept of *management by subscription*, which is the concept of managing network resources by creating sets of profiles and distributing the profiles (through *profile managers*) to physical entities called subscribers. A profile is a container for application-specific (in this case, Tivoli Inventory-specific) information and a profile manager is a container for a set of profiles that have subscribers, or systems targeted for distribution in common.

Valid targets for an inventory profile distribution are profile managers and *endpoints*. An endpoint is a system that is managed from another system, called a gateway. It is used as a target of an operation such as receiving a profile or executing a task.

To use Tivoli Inventory, you must first create and customize at least one inventory profile within a profile manager. From the profile manager window, distribute the inventory profile to subscribers. The profile then performs an action on each target system. It can distribute a configuration file to the system, run a scan on the system, or both, depending on the profile settings.

The scanner creates *MIF (Management Information Format)* files. MIF files are ASCII files that contain information in a standard format. These files are collected during hardware, software, or custom scans. Next, Tivoli Inventory processes the information in MIF files and uses SCS or MDist 2 to send the data to the Tivoli Inventory configuration repository.

After Tivoli Inventory updates the configuration repository, you can view the information with the Tivoli Management Framework query feature. For example, you can query the configuration repository for all systems that have an outdated version of a software product that will need upgrading in the next year.

Tivoli Inventory also works with SCS to control when and how much data is collected on your network.

Using Tivoli Inventory with Scalable Collection Service

This section describes SCS, lists the components of an SCS environment, describes how SCS and Tivoli Inventory work together to collect data, and lists the features and advantages of SCS.

Scalable Collection

Scalable collection is the ability to control how much data is collected on your network as well as when the data is collected. With scalable collection, you can significantly reduce the time needed to scan and store vast amounts of data. SCS gives you the ability to scale data collection.

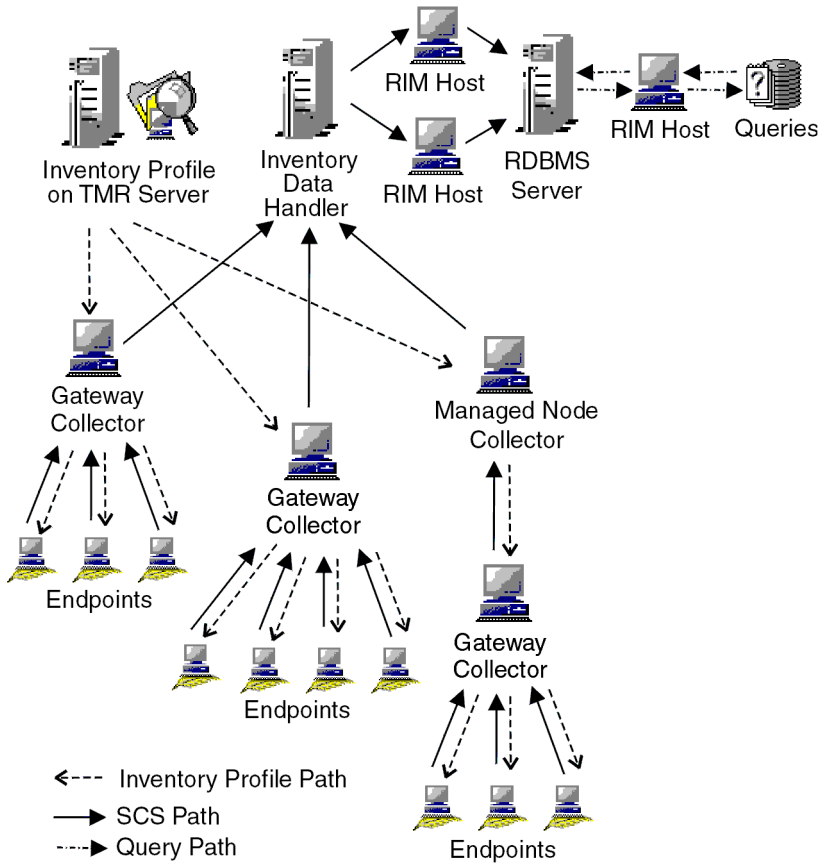
SCS sends scan data to its destination as the scan on each endpoint completes, so it sends smaller chunks of data through the network. Another advantage of SCS is its ability to stage data through a hierarchy of collector nodes, which distributes the processing load across the TMR. In essence, SCS more effectively manages the stream of data while creating multiple pathways to the configuration repository. This functionality results in faster scans and increased control of network resources.

SCS is triggered by an endpoint that has data to be collected. This endpoint informs the gateway that the data is ready by sending the CTOC. A collector daemon running on the gateway then retrieves the data from the endpoint and stores it on the collector in persistent storage. Depending on the configuration of your network, the data may then travel through a hierarchy of collectors before arriving at its destination.

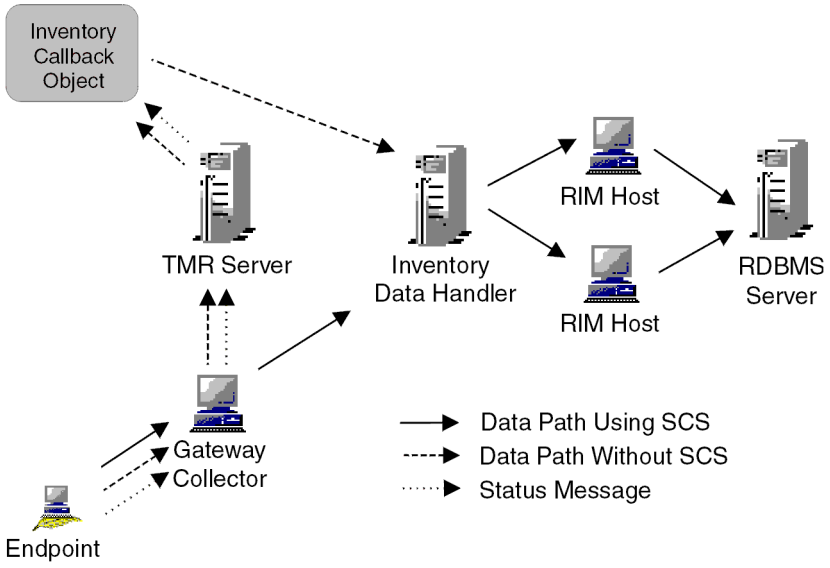
With Tivoli Inventory, all collectors send data to the inventory data handler, which then sends the data to the configuration repository.

Data Collection with SCS

The following diagram illustrates the way SCS and Tivoli Inventory work together to complete scans and store information in the configuration repository.



The following diagram shows the role of the inventory callback object in data collection.



Data collection using Tivoli Inventory and SCS occurs in three major phases:

- First, the inventory profile scans endpoints, which causes the endpoints to send CTOCs to SCS.
- Next, SCS processes the CTOCs and moves data through the collector hierarchy to the inventory data handler.
- Finally, the inventory data handler writes the data to one or more RIM objects, which send the data to the configuration repository.

These phases are described in the following paragraphs.

In the first phase, you distribute the profile to the endpoints to be scanned. As each scan completes on each endpoint, Tivoli Inventory generates a compressed and encoded data file. The endpoint sends a CTOC, which contains information about the data file, to the collector on the gateway that manages the endpoint. The collector daemon queues the CTOC for processing.

In the second phase, SCS moves data from each endpoint to a depot in the gateway collector that controls the endpoint. While the data moves

from the endpoint to the depot, the CTOC for the data remains in the input queue of the collector. When the collector has collected all the data from the endpoint, the CTOC moves to the output queue. The collector then notifies the next collector in the hierarchy that the data is ready to be collected. The upstream collector places the CTOC in its input queue, and then collects the data from the downstream collector. When the data is completely transferred to the upstream collector, the downstream collector discards the CTOC in its output queue and data in its depot.

This same exchange happens between each collector in the hierarchy as the collectors asynchronously transfer the data to the inventory data handler.

In the third phase, the inventory data handler sends the collected data to any available RIM object. From the RIM object, the collected data is sent to the Tivoli Inventory configuration repository. The inventory data handler sends the completion status of each scanned endpoint, as well as any error information, to the status collector. The status collector changes the status for each endpoint from pending to either successful or failed.

Tivoli Inventory and SCS Features

In addition to gathering information about systems in your enterprise, Tivoli Inventory provides a central point of management for the systems in a distributed client/server environment. The following sections describe the features of Tivoli Inventory and SCS.

Tivoli Inventory Features

Tivoli Inventory offers the following features:

- Flexible, standardized data storage architecture—Tivoli Inventory uses *Desktop Management Task Force (DMTF)* 2.0 MIF files to hold all the information gathered during an inventory scan. Tivoli Inventory also stores the contents of MIF files in the configuration repository.

  Note: DMTF is a consortium whose goal is to define standard syntax for data that describes how to manage the components of desktop computers.

- Extensive customization options for the inventory profile—you can configure an inventory profile to do any of the following during a distribution to target systems:
  - Scan hardware and software—Tivoli Inventory can scan target systems for hardware information, software information, or both. You can customize a software scan to scan particular directories and files.

    On PCs, hardware scans automatically scan for BIOS information in addition to other hardware components. Also, you can choose to scan for Desktop Management Information (DMI).
  - Run a script—you can include a script with an inventory profile that runs on target systems during a profile distribution. Scripts can, among other things, run custom scanners which collect information and create custom MIF files from that information.
  - Read hardware, software, or custom MIF files—Tivoli Inventory can read the MIF files that are created by either a hardware or software scan or the MIF files that you create.

  See Chapter 5, “Working with Inventory Profiles” and Chapter 6, “Distributing Inventory Profiles” for more information about creating, customizing, and distributing inventory profiles.
- Configurable hardware scanning—You can select the type of information to be returned from a hardware scan and reduce the amount of data generated by the scan. This reduction in data can improve the speed with which the scan runs, reduce the network bandwidth required to transport the scan data, and reduce the amount of time to write the data into the configuration repository.
- Windows Management Instrumentation (WMI) scanner—Tivoli Inventory can get data from endpoints using WMI to receive information stored in the Common Information Model (CIM) on Windows platforms supporting CIM and WMI for common inventory attributes. This feature is used automatically when it is available.

- Endpoint-initiated scans—You can initiate a scan from an endpoint instead of distributing the scan from an inventory profile. This feature has many uses. For example, you can use endpoint-initiated scans to perform the following actions:
  - Scan a machine each time it is rebooted
  - Scan a laptop system each time a mobile user logs in to the network
  - Scan a machine that is disconnected from the network
- An Inventory Query dialog box through which you can search the configuration repository for a single system that meets your criteria. See Chapter 8, “Querying Inventory Information” for more information.
- History tracking—You can create history tables that record any changes to the scan data in the Tivoli Inventory configuration repository. If you do not create these tables, Tivoli Inventory does not track any changes to scan data. See “History Tracking” on page C-2 for more information.
- A Web-based interface—using a Web browser or the command line, you can use the User Data Template to create the User Data Form for users. With the User Data Form, you can create MIF files containing custom information that can later be collected during a scan. For example, users can enter their office and phone numbers.

  See Chapter 7, “Collecting Custom Information with Tivoli Inventory” for more information about Tivoli Inventory Web pages.
- Mobile endpoint support—You can create a profile for a customized inventory scan that can be pushed to a mobile or remote endpoint. The scan can be scheduled to run when the endpoint is disconnected; the data is sent the next time the endpoint is connected to the network.
- Wake-on-LAN technology—With MDist 2 functionality, Tivoli Inventory can automatically wake up systems that have a wake-on-LAN-enabled network card using the Intel Wired for Management specification.

- Integration with other Tivoli applications—you can use Tivoli Inventory in conjunction with the Tivoli Management Framework query facility and Tivoli Software Distribution to perform system management tasks. For more information, see "Tivoli Inventory and Other Tivoli Applications" on page 1-14.

SCS Features

The following SCS features illustrate the advantages of asynchronous data collection:

- Asynchronous data collection and asynchronous processing lead to better distribution of the processing load across more nodes in the TMR, as well as better use of the network. SCS stages the data at different collectors in the network as it flows to the configuration repository, and allows control over the data flow between successive collectors in the hierarchy.
- SCS returns scan data as the scan of each endpoint completes. This feature reduces RIM object overload at the end of a distribution.
- After data has been collected from an endpoint, that endpoint can be disconnected without affecting SCS.
- The inventory data handler can write data in parallel to one or more RIM objects, each of which has one or more connections.
- SCS reduces the load on the TMR server. The inventory data handler stages scan data for individual endpoints while the configuration repository is updated. (It is recommended that you create the inventory data handler on a system other than the TMR server.)
- Using SCS and the Tivoli scheduler, you can schedule scans and data collection traffic for times when network traffic is at a minimum.
- Collectors in the collector hierarchy store collected data in depots. If a failure occurs, the collector can resend the data when it is back online, so you do not have to scan all the endpoints again.
- SCS provides functionality through which you can get information about the status of collectors, CTOCs, and scans.

- With SCS, you can determine the status of individual targets as they complete, rather than having to wait until all targets complete.
- SCS allows you to retry collecting data from endpoints or other collectors following a failure; you can set the specific number of retries.

See Chapter 4, "Working with Scalable Collection Service," for more information about using the features of SCS.

Tivoli Inventory and Other Tivoli Applications

Tivoli Inventory contributes to the Tivoli approach of *full-cycle deployment*. You can use single-action management to track unauthorized changes in software configurations and re-establish the appropriate software configuration by using a combination of Tivoli Inventory, Tivoli Software Distribution, and Tivoli Management Framework tasks. Combining applications and tools enables you to perform many useful tasks, including the following operations:

- Using Tivoli Inventory and the Tivoli Management Framework to automatically notify an administrator if unauthorized applications are added to any of the systems in your enterprise.
- Using Tivoli Inventory, Tivoli Software Distribution, and the Tivoli Management Framework to check the system software configurations to see if any critical files are missing, then re-establish the proper configuration. After creating and deploying management-ready applications, you can continually synchronize applications and system configurations on an enterprise scale.
- Using Tivoli Inventory and the Tivoli Management Framework query facility to track and report on system hardware and software for accounting purposes.

Security

Tivoli Inventory functions within the bounds of Tivoli Management Framework security to ensure secure and authorized use of the application. In other words, each administrator who wants to work with Tivoli Inventory must be given the proper Tivoli security authorization, or role, to complete inventory tasks. For more information about

assigning roles, see the *Tivoli Management Framework User's Guide*.

Many procedures in this guide are prefaced with a table showing the role required for the procedure. See Appendix A, "Authorization Roles" for more information about the inventory authorization roles required for each operation.

2

Installing Tivoli Inventory, Version 4.0

This chapter provides the steps necessary to install Tivoli Inventory, Version 4.0, in your Tivoli environment.

Overview

Before installing Tivoli Inventory, complete the following steps:

1. Read the *Tivoli Inventory Release Notes, Version 4.0*.
2. Install and configure the 3.7.1 Version of the Tivoli Management Framework on the Tivoli Management Region (TMR) server. Ensure that each system you want to scan is an endpoint in the TMR. See the *Tivoli Management Framework Planning for Deployment Guide*, *Tivoli Management Framework User's Guide*, and the *Tivoli Management Framework Release Notes* for more information.
3. Ensure that a Web browser is installed on any system from which the UserLink for Tivoli Inventory features will be used. Consult the Web browser documentation for installation instructions. See the *Tivoli Inventory Release Notes* for a list of supported Web browsers.
4. Back up the Tivoli Management Framework object database for all systems in the TMR.

The following steps provide an overview of the process of installing and setting up Tivoli Inventory, Version 4.0.

1. Install Scalable Collection Service (SCS). This service installs as a patch to Tivoli Management Framework. For more information, see "Installing Scalable Collection Service" on page 2-4.
2. Install and configure an RDBMS to communicate with the RIM host. *You must install an RDBMS before you install Tivoli Inventory*. See "Configuring an RDBMS" on page 2-11. For a list of supported RDBMSs, see the *Tivoli Management Framework Release Notes, Version 3.7.1*.
3. Install the Tivoli Inventory Server application on the TMR server and managed nodes. See "Installing Tivoli Inventory Server on the TMR Server and Managed Nodes" on page 2-21.
4. Install the Tivoli Inventory Gateway application on managed nodes that you have configured in the Tivoli Management Framework as gateways. See "Installing Tivoli Inventory Gateway" on page 2-31.

This chapter also includes information about removing an installation of Tivoli Inventory, Version 4.0. See "Removing Tivoli Inventory" on page 2-33.

After you have installed the Tivoli Inventory products on the systems in your TMRs, you must complete the following steps to be able to store and retrieve the information that Tivoli Inventory collects. Chapter 3, "Creating the Configuration Repository" covers the following tasks:

1. Create the configuration repository. See Chapter 3, "Creating the Configuration Repository."
2. Install the provided list of software signatures in the configuration repository. See "Installing Software Signatures" on page 3-21.
3. Create the Tivoli Inventory queries. See "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Coexistence with Previous Product Versions

Tivoli Inventory, Version 4.0, is a new product and not an upgrade of Tivoli Inventory, Version 3.6.*x*. While this new product does not require

the installation of previous versions of Tivoli Inventory, it can coexist with previous versions. You might choose to install both the new version and a previous version of Tivoli Inventory for the following reasons:

- To ease the migration from a previous version to the new version.
- To keep your current, customized environment operational while beginning to exploit the new features offered by Tivoli Inventory, Version 4.0.

See the See the *Tivoli Inventory Release Notes* for more information about configuring Tivoli Inventory, Version 4.0, to operate with previous versions.

Using Tivoli Software Installation Service

Tivoli Software Installation Service (SIS) is a product that can simultaneously install multiple Tivoli products on multiple systems. This Java-based product can, therefore, install more products on more systems in much less time than the Tivoli Management Framework install facility. SIS performs product prerequisite checks and, if defined, user-specified prerequisite checks, ensuring as few installation failures as possible. In most cases, failures occur only when systems are turned off or removed from the network.

SIS also creates an install repository (IR) into which you can import the installation image of one or more Tivoli products. You can choose to import only those interpreter types needed in your environment, which saves disk space and import time. The IR is the source of all your Tivoli Management Framework installations. You can even share a single IR across multiple TMRs.

Tivoli recommends that you upgrade the install facility in your current Tivoli Management Framework installation by installing SIS. If you are installing Tivoli Management Framework for the first time, install SIS on the first managed node running an SIS-supported operating system. Once installed, use SIS to install other Tivoli products.

See the *Tivoli Software Installation Service User's Guide* for instructions on how to install SIS in your Tivoli Management Framework installation and how to install products using SIS.

Installing Scalable Collection Service

The Scalable Collection Service (SCS) extends the capability of the Tivoli managed node and associated gateways. SCS installs as a patch to the Tivoli Management Framework product. It is recommended that you install SCS on all supported managed nodes in each Tivoli Management Region (TMR).

SCS uses a collector hierarchy to collect, stage, and send data to its destination. The collector hierarchy is based on the repeater hierarchy established by the multiplexed distribution (MDist 2) service.

There are two circumstances under which you might be installing SCS:

- In the first case, you are installing Tivoli products for the first time and have not established a repeater hierarchy.
- In the second case, your repeater hierarchy is already in place.

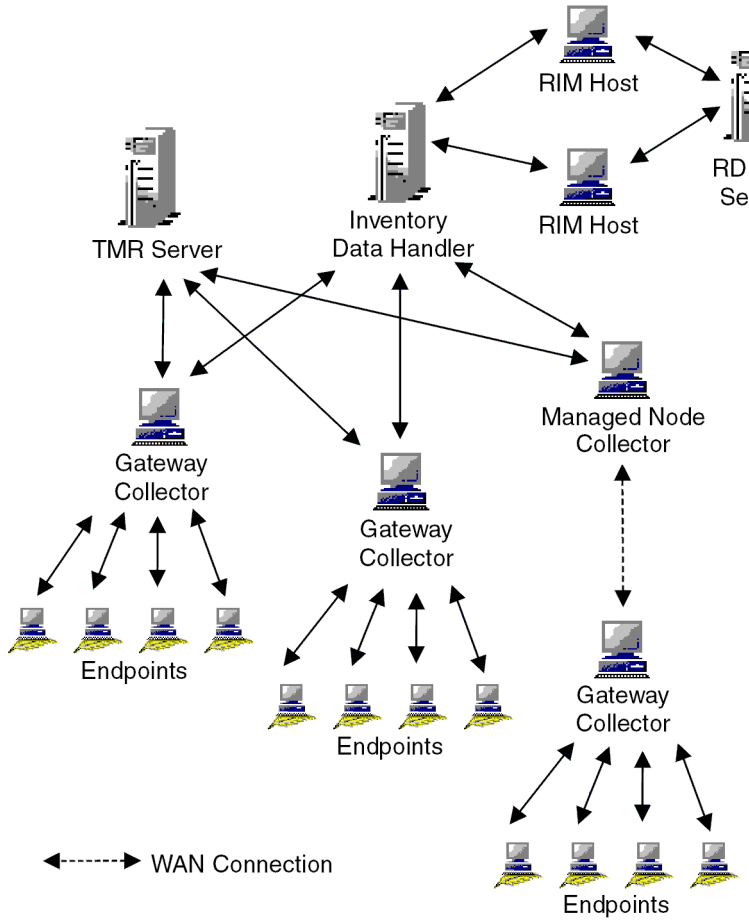
If you have not established a repeater hierarchy, SCS uses a default configuration in which each gateway is a collector that sends data directly to the inventory data handler.

If you have already established a repeater hierarchy, SCS mirrors the repeater hierarchy. When you add a new repeater to your hierarchy, you must install SCS on the managed node that contains the new repeater if you have not already done so.

Because the SCS and MDist 2 services share the same network topology, you should plan your hierarchy of collectors and repeaters carefully. For example, any changes that you make to the repeater hierarchy are also made to your collection hierarchy and vice versa. You should therefore set up your network in a way that is optimal for both the SCS and MDist 2 services.

Note: When you change your repeater hierarchy, your collector hierarchy does not change until you run the **wcollect** command using the **–r** option on all affected collectors, or reexecute the TMR server and all systems acting as collectors using the **odadmin** command. For more information about the **wcollect** command, see “wcollect” on page A-6. For more information about the **odadmin** command, see the *Tivoli Management Framework Reference Manual*.

It is usually beneficial to place a system configured to be a collector and repeater on either end of a critical link in your network, such as a wide area network (WAN) connection. With this configuration, data has to be collected from only one system across the WAN. The following figure illustrates this configuration.



For more information about setting up a repeater hierarchy, see the *Tivoli Management Framework Planning for Deployment Guide*.

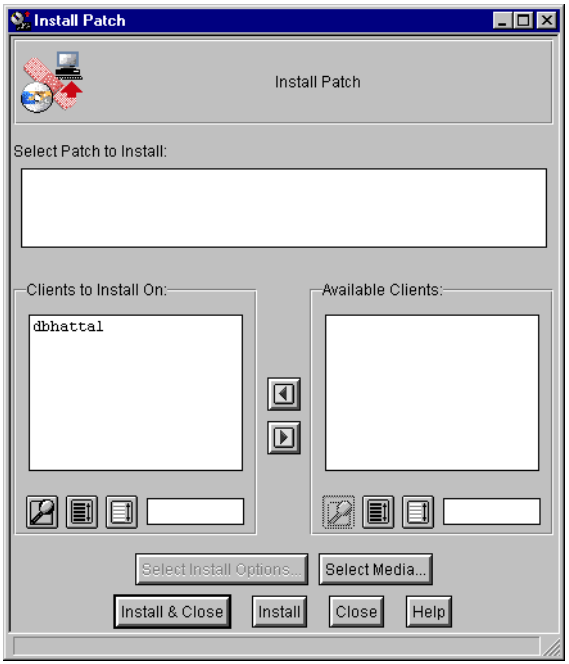
Installing the SCS Patch

The following sections provide the procedures to install SCS to work with Tivoli Inventory. You must install SCS on all managed nodes in your Tivoli environment.

Selecting the Media

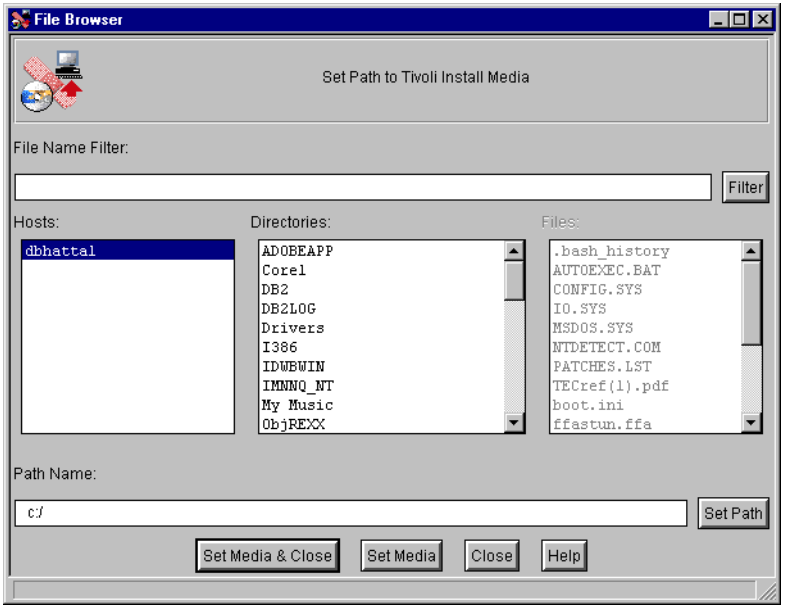
Complete the following steps to obtain access from the Tivoli desktop to the files that you need to install:

1. Run the Tivoli desktop on the TMR server.
2. Select **Install –> Install Patch** from the Desktop menu. The Install Patch dialog box is displayed.



If Tivoli Scalable Collection Service For Framework 3.7.1 is already in the Select Patch to Install scrolling list, go to “Installing Tivoli Scalable Collection Service For Framework 3.7.1” on page 2-8. Otherwise, continue with step 3.

3. Click **Select Media**. The File Browser dialog box is displayed.



4. Enter the location of the Tivoli CD-ROM in the Path Name text box by completing one of the following tasks:

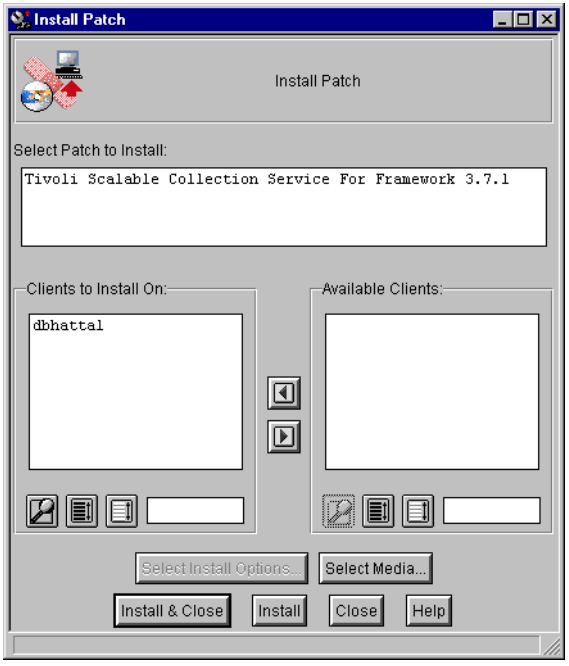
   - Type the complete path name in the Path Name text box.
   - Browse the file system by completing the following steps:

     a. In the Hosts scrolling list, select the host (or drive) on which the CD-ROM is mounted. Choosing a host updates the Directories scrolling list to show the directories (under root) of the host you selected.

     b. In the Directories scrolling list, double-click the directory that contains the installation media. Choosing a directory updates the Files list.

     c. Click **Set Path**.

5. Click **Set Media & Close** and return to the Install Patch dialog box. Tivoli Scalable Collection Service For Framework 3.7.1 is displayed in the Select Patch to Install scrolling list.

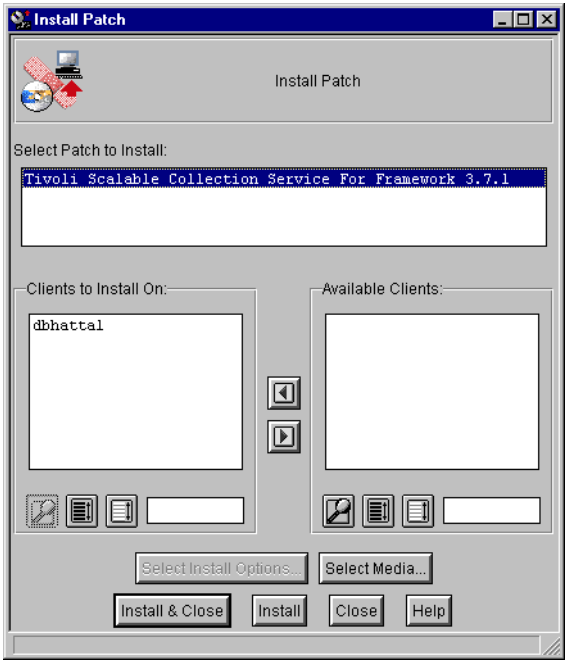


Installing Tivoli Scalable Collection Service For Framework 3.7.1

Complete the following steps to install the SCS patch called Tivoli Scalable Collection Service For Framework 3.7.1 on managed nodes that will act as gateways for managing endpoints, managed nodes that will serve as collectors, and the TMR server:

1. Select **Install –> Install Patch** from the Desktop menu. The Install Patch dialog box is displayed.

2. In the Install Patch dialog box, select **Tivoli Scalable Collection Service For Framework 3.7.1** from the Select Patch to Install scrolling list.



3. Make sure that the TMR server and all managed nodes on which you want to install SCS appear in the Clients to Install On scrolling list.

   Note: It is recommended that you install SCS on all supported managed nodes. You cannot install SCS on OS/390 or OS/2 systems.

4. In the Clients to Install On scrolling list, select any managed nodes on which you do *not* want to install SCS. Click the right-arrow button to move the selected managed nodes to the Available Clients scrolling list.

5. To begin installing SCS, click **Install**. The Patch Install dialog box provides a list of the operations that take place and warns you of

any problems that you might want to correct before installing. You can choose one of the following options:

- Review the status information and click **Continue Install**. The Patch Install dialog box informs you when installation is finished.
- To install SCS at another time, click **Cancel**.

Command Line

You can use the **wpatch** command to install SCS from the command line. The following example installs SCS on the managed nodes x, y, and z.

```
wpatch -c $IMAGE_PATH/cdrom -i MCOLLECT.IND x y z
```

For more information about the **wpatch** command, see the *Tivoli Management Framework Reference Manual*.

Using RIM and the RDBMS

Tivoli Inventory scans target systems and stores the information in the configuration repository, which is an RDBMS database. The configuration repository holds information about the configuration of systems in your Tivoli environment. Tivoli Inventory uses the RDBMS Interface Module (RIM) to communicate with the RDBMS server.

Tivoli Inventory stores information in the configuration repository, which is installed with Tivoli Inventory. Tivoli Software Distribution also stores information in the configuration repository. If you are also using Tivoli Software Distribution, see the *Tivoli Software Distribution User's Guide* for more information about how Tivoli Software Distribution uses the configuration repository.

The RDBMS cannot work with Tivoli Inventory unless you configure the RDBMS client and server software, which enables the RIM host to communicate with the RDBMS.

Before you begin the Tivoli Inventory installation, you must select a RIM host and an RDBMS server. Tivoli recommends that you use different systems for the TMR server, the RDBMS server, and the RIM host.

Choosing a RIM Host

The RIM host is the managed node on which the RIM objects are installed. You designate the RIM host by entering its name in the RIM Host text box on the Install Options dialog box, which is displayed when you install Tivoli Inventory. For a description of the RIM objects that Tivoli Inventory creates during installation, see "Changing Passwords for RIM Objects" on page 2-28.

Consider the following when you choose your RIM host:

- It must be a managed node in the TMR.
- It must have the RDBMS client software installed. See "Configuring an RDBMS" on page 2-11 for more information.
- It communicates with the RDBMS.
- If the RIM host is an HP-UX or Windows NT system, ensure that the user tmersrvd is defined on that system. See the *Tivoli Management Framework Release Notes* for more information about the user tmersrvd.

Choosing an RDBMS Server

The RDBMS server is a system on which you install the RDBMS and the Tivoli Inventory configuration repository schema. Consider the following requirements when you choose your RDBMS server:

- It must have a TCP/IP connection to the RIM host.
- It does not have to be a managed node in the TMR, but it must be on the same network as the TMR.
- It must have enough disk space to support the amount of information you plan to store in the configuration repository. Space requirements are included in the *Tivoli Inventory Release Notes*.
- Multiple TMRs and applications can use it.

Configuring an RDBMS

When you install Tivoli Inventory, you are asked to enter RDBMS configuration information in the Install Options dialog box. This

information is used to create the RIM host in the Tivoli Management Framework object database, which enables Tivoli Inventory to communicate with the RDBMS. For this reason, you must install and configure an RDBMS before you install Tivoli Inventory.

After configuring an RDBMS and installing Tivoli Inventory on your TMR, install the Tivoli Inventory configuration repository schema by running the RDBMS scripts, which are provided with Tivoli Inventory. See Chapter 3, "Creating the Configuration Repository," for more information.

If you scan interconnected TMRs and store scan results in a single RDBMS database, ensure that the text boxes on the Install Options dialog box are set to the same values during the Tivoli Inventory installation for each TMR. You can use the **wgetrim** command to check the settings of the RIM host in a TMR.

If the settings are not the same, use the **wsetrim** and **wsetrimpw** commands to change the settings. Once the TMRs are connected, you can perform scans across TMR connections. For more information about RIM commands, see the *Tivoli Management Framework Reference Manual*.

The following sections include instructions for configuring each supported RDBMS.

Configuring DB2

Install a DB2 RDBMS and create a DB2 instance and database on the RDBMS server before you install Tivoli Inventory. You must create the database as the instance owner. Consult your DB2 documentation for instructions about creating a DB2 instance and database. The database you create becomes the Tivoli Inventory configuration repository after you run the scripts to install the Tivoli Inventory schema. See "Creating a DB2 Configuration Repository" on page 3-2 for more information.

Select a managed node to be the RIM host. See "Choosing a RIM Host" on page 2-11 for more information about selecting a RIM host.

Tivoli recommends that you use separate systems for your RDBMS server and RIM host. Because the RDBMS server must communicate with the RIM host, the RIM host must have a DB2 client instance and database installed. Also, your DB2 client instance user name must be the

same user name as the server instance owner. However, the client instance user name for Tivoli Inventory, Version 4.0, should not be the same as the Tivoli Inventory, Version 3.6.2, user name (if you had one). The default user name in Tivoli Inventory, Version 4.0, is invtiv. Complete the steps below to install a client instance and database on the RIM host.

1. Log in to the RIM host as the user that will be the instance owner. The instance owner is authorized to access the Tivoli Inventory configuration repository.

   Note: After you install Tivoli Inventory and create the configuration repository, use the **wsetrimpw** command to change the password from tivoli, the default, to the instance owner's password.

2. On the RIM host, start the DB2 interactive SQL utility. Enter the following command:

   ```
   db2
   ```

3. Create a new client node that can connect to the RDBMS server. Enter the following command:

   ```
   catalog tcpip node nodename remote hostname \
   server servicename
   ```

   where:

   nodename

   Specifies the name of the system on which the client database resides. This name must be unique in your node directory list.

   hostname

   Specifies the name of the RDBMS server.

   servicename

   Specifies the server instance entry. The server instance entry is located in the /etc/services file for UNIX and the \winnt\system32\drivers\etc\services file for Windows NT. There must be a server instance

entry on both the RIM host and the RDBMS server.

4. Verify that the node was created. Enter the following command:

   ```
   list node directory
   ```

   The name, host name, and service name of the node you created is displayed.

5. Catalog the server database at the client instance on the RIM host. Complete the following steps:

   a. To catalog the database, enter the following command:

      ```
      catalog database databasename as aliasname at node \
      nodename
      ```

      where:

      databasename

      Specifies the name of the database on the RDBMS server that you created for Tivoli Inventory.

      aliasname

      Specifies either the alias name for the client instance or the same database name used at the server.

      nodename

      Specifies the name of the system on which the client instance that you created resides.

   b. Verify that the database is cataloged. Enter the following command:

      ```
      list database directory
      ```

      The name of the alias, its database, and the node on which the server database resides is displayed.

Configuring Informix

Install the Informix RDBMS and create a Informix instance and database on the RDBMS server before you install Tivoli Inventory. You

must create the database as the user informix. Consult your Informix documentation for instructions about creating an Informix instance and database. The database that you create becomes the Tivoli Inventory configuration repository after you run the scripts to install the Tivoli Inventory schema. See "Creating an Informix Configuration Repository" on page 3-7 for more information.

Select a managed node to be the RIM host. See "Choosing a RIM Host" on page 2-11 for more information about selecting a RIM host.

Tivoli recommends that your RDBMS server and RIM host should be different systems. Because the RDBMS server must communicate with the RIM host, the RIM host must have Informix client software installed.

Configuring MS SQL Server

Install MS SQL Server on the RDBMS server. Consult your MS SQL Server documentation for installation instructions.

CAUTION:
The Installation Options dialog that is displayed during MS SQL Server installation includes a Sort Order option. To be able to use MS SQL Server for your Tivoli Inventory configuration repository, you must select the Dictionary order, case-sensitive option at installation.

Select a Windows NT managed node to be the RIM host. See "Choosing a RIM Host" on page 2-11 for more information about selecting the RIM host. After you install Tivoli Inventory, you can run the scripts that create the database and install the configuration respository schema. See "Creating an MS SQL Server Configuration Repository" on page 3-9 for more information.

Tivoli recommends that you use separate systems for your RDBMS server and RIM host; therefore, you must install MS SQL Server or client on the RIM host.

Configuring Oracle

Install an Oracle RDBMS and SQL*Net on the RDBMS server. Consult your Oracle documentation for installation instructions. Select a managed node to be the RIM host. See "Choosing a RIM Host" on page 2-11 for more information. After you install Tivoli Inventory, you can run the scripts that create the database and install the configuration

respository schema. See “Creating an Oracle Configuration Repository” on page 3-13 for more information.

Tivoli recommends that you use separate systems for your RDBMS server and RIM host. Complete the following steps to configure the RIM host as an Oracle client using SQL*Net. Consult your Oracle documentation if you want to use Oracle Names:

1. Install Oracle client software, which includes SQL* Plus, on the RIM host.
2. Copy the tnsnames.ora file from the $ORACLE_HOME/network/admin directory on the RDBMS server to the $ORACLE_HOME/network/admin directory on the RIM host, where $ORACLE_HOME is the path to the directory where your Oracle server or client installation resides.
3. Verify that the tnsnames.ora file is properly updated to reflect your configuration. In other words, check that the host name of the server, the Oracle instance ID, the port sqlnet is connected to, and the communication protocol are accurate.

Configuring Sybase

Install the Sybase RDBMS. Consult your Sybase documentation for instructions. Select a managed node to be the RIM host. See “Choosing a RIM Host” on page 2-11 for more information. After you install Tivoli Inventory, you can run the scripts that create the database and install the configuration respository schema. See “Creating a Sybase Configuration Repository” on page 3-17 for more information.

Tivoli recommends that you use separate systems for your RDBMS server and RIM host. Complete the following steps to configure the RIM host as a client of the RDBMS server:

1. Install the Sybase client software, which includes isql, on the RIM host.
2. If the RIM host is a Windows NT system, ensure that the PATH variable includes the directory where the Sybase DLLs are installed.
3. Copy the interfaces file from the RDBMS server to the directory on the RIM host where the Sybase client software is installed.

If your RIM host is a Windows NT system, make sure that the SQL.INI file exists in the $SYBASE/INI directory on the RIM host, where $SYBASE is the path to the directory where your server or client Sybase installation resides.

Note: If the RDBMS server is a Solaris system and your RIM host is not or vice versa, do not copy the interfaces file to your RIM host. The interfaces file for Solaris is not compatible with other operating systems. You must create a new interfaces file that is compatible. Consult your database administrator for assistance.

Preparing to Install Tivoli Inventory Products on Target Systems

After you choose a RIM host and install and configure your RDBMS server, you must install Tivoli Inventory Server or Tivoli Inventory Gateway on the appropriate systems in the TMR.

Before and after installing Tivoli Inventory, it is recommended that you back up your Tivoli Management Framework object database for all systems in the TMR. This makes it possible to go back to the state of the object database before installation. This is useful if, for some reason, you encounter a problem while installing Tivoli Inventory. From the Tivoli desktop, select **Backup** from the Desktop menu to perform a backup of the Tivoli server and clients. You can also use the **wbkupdb** command. See the *Tivoli Management Framework Reference Manual* for information about the **wbkupdb** command.

The following table provides the authorization roles required to perform the Tivoli Inventory installation:

Operation	Context	Role
Install Tivoli Inventory products	Tivoli desktop	**Install_product** or **senior**

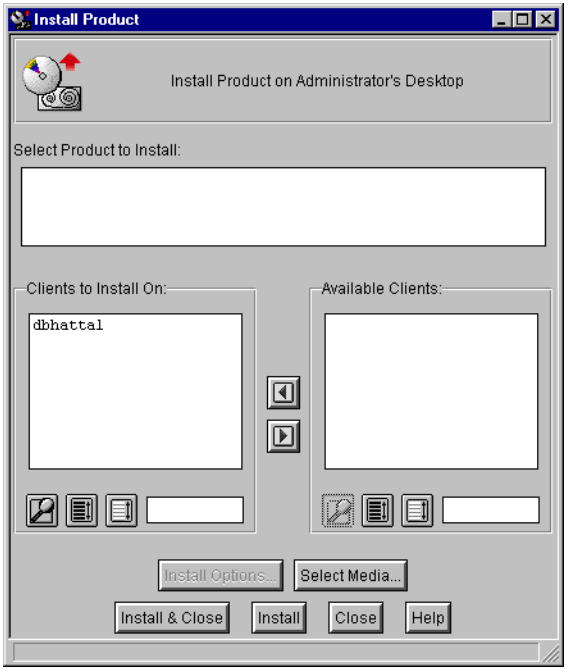
You can install Tivoli Inventory, Version 4.0 from either the Tivoli desktop or the command line. The process of installing Tivoli Inventory on target systems can be divided into the following tasks:

1. “Selecting the Media” on page 2-18
2. “Choosing Between Tivoli Inventory Server and Tivoli Inventory Gateway” on page 2-21
3. “Installing Tivoli Inventory Server on the TMR Server and Managed Nodes” on page 2-21
4. “Installing Tivoli Inventory Gateway” on page 2-31

Selecting the Media

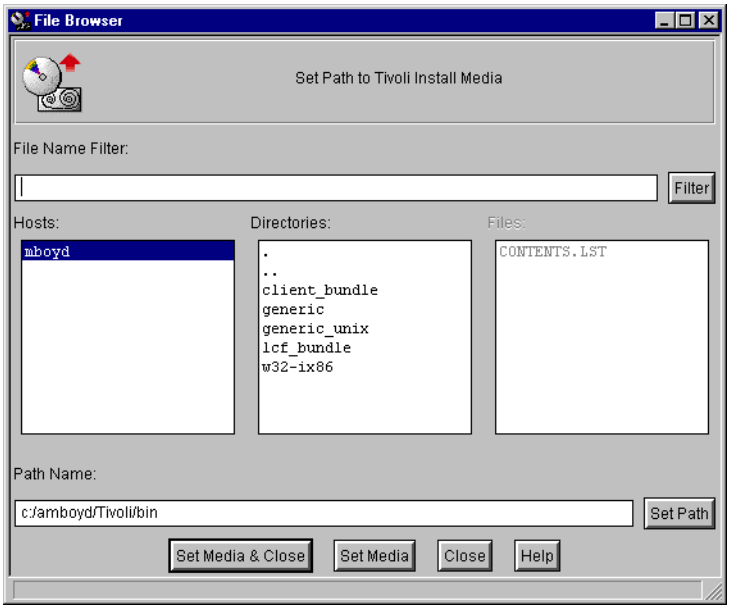
Complete the following steps to select the drive on which the Tivoli CD-ROM is installed and to obtain access from the Tivoli desktop to the Tivoli Inventory files that you need to install:

1. Run the Tivoli desktop on the TMR server.
2. Select **Install –> Install Product** from the Desktop menu. The Install Product dialog box is displayed.



If the Tivoli Inventory products are already listed in the Select Product to Install scrolling list, go to "Installing Tivoli Inventory Server on the TMR Server and Managed Nodes" on page 2-21. Otherwise, continue with step 3.

3. Click **Select Media**. The File Browser dialog box is displayed.



4. Enter the location of the Tivoli CD-ROM in the Path Name text box by completing one of the following tasks:

   - Type the complete path name in the Path Name text box.
   - Browse the file system by completing the following steps:

     a. In the Hosts scrolling list, select the host (or drive) on which the CD-ROM is mounted. Choosing a host updates the Directories scrolling list to show the directories (under root) of the host you selected.

     b. In the Directories scrolling list, double-click the directory that contains the installation media. Choosing a directory updates the Files list.

     c. Click **Set Path**.

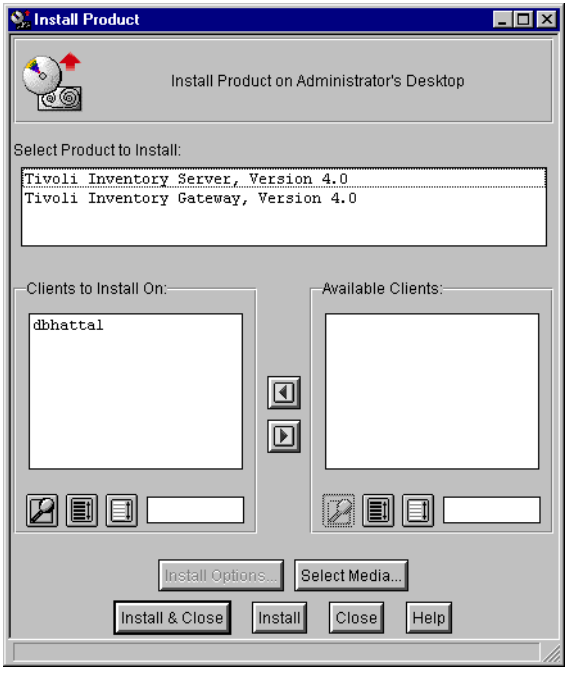
5. Click **Set Media & Close** and return to the Install Product dialog box. The following products that are available for installation are displayed in the Select Product to Install scrolling list:

   - Tivoli Inventory Server, Version 4.0

     Includes the files that enable you to create and distribute profiles and run Tivoli Inventory commands. Install this product on the TMR server and the managed nodes from which you want to manage systems in your TMR.

   - Tivoli Inventory Gateway, Version 4.0

     Includes the files that enable a gateway to perform Tivoli Inventory functions on endpoints. Install this product on all managed nodes that you have configured as gateways.



Choosing Between Tivoli Inventory Server and Tivoli Inventory Gateway

On managed nodes, you can install both Tivoli Inventory Server and Tivoli Inventory Gateway. However, you can use the following guidelines when you plan your installation to save time and memory:

- Install only Tivoli Inventory Server on managed nodes that you want to use to create and distribute inventory profiles and run Tivoli Inventory commands. See "Installing Tivoli Inventory Server on the TMR Server and Managed Nodes" on page 2-21 for the procedures.
- Install only Tivoli Inventory Gateway on systems that you want to use as endpoint gateways. See "Installing Tivoli Inventory Gateway" on page 2-31 for the procedures.
- Install both Tivoli Inventory Server and Tivoli Inventory Gateway on managed nodes that you want to use for both functions.

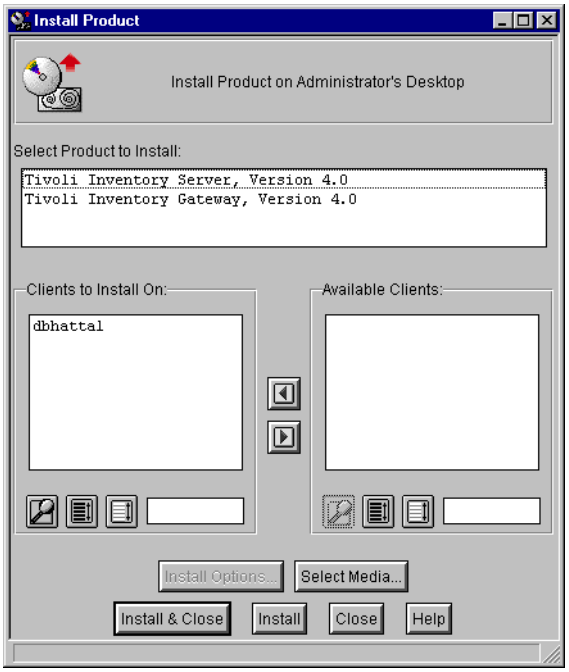
For more information about installing managed nodes and gateways, see the *Tivoli Management Framework Planning for Deployment Guide*.

Installing Tivoli Inventory Server on the TMR Server and Managed Nodes

Once you have selected the media, complete the following steps to install Tivoli Inventory Server, Version 4.0 and man pages for Tivoli Inventory commands on the TMR server and the managed nodes from which you want to create and distribute profiles or run Tivoli Inventory commands. You can complete this task from the desktop or from the command line.

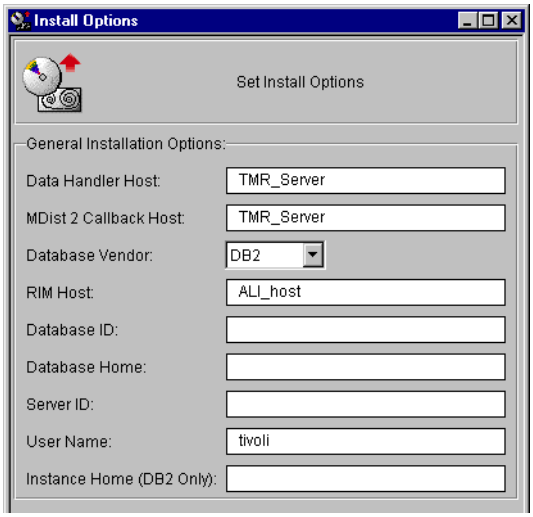
Desktop

1. Select **Install –> Install Product** from the Desktop menu. The Install Product dialog box is displayed.



2. In the Install Product dialog box, select **Tivoli Inventory Server, Version 4.0** from the Select Product to Install scrolling list.

   The Install Options dialog box is displayed. This dialog box requires information that is specific to your RDBMS and RIM host.



3. Complete the Install Options dialog box. The following steps describe in general the required information. For details specific to your RDBMS, see the appropriate RDBMS section at the end of this step.

   a. In the Data Handler Host text box, enter the name of the managed node that you want to serve as the inventory data handler. It is recommended that you do *not* put the inventory data handler on the same managed node as the TMR server.

   b. In the MDist 2 Callback Host text box, enter the name of the managed node that you have configured to be the TMR server. This step specifies the managed node on which the inventory callback object is installed.

   c. From the Database Vendor menu, select the vendor name of the RDBMS you are using with Tivoli Inventory.

   d. In the RIM Host text box, enter the name of the managed node that you have configured to be the RIM host for the TMR. If you have not already chosen a RIM host, see "Choosing a RIM Host" on page 2-11.

   e. In the Database ID text box, enter the name of the Tivoli Inventory database in the RDBMS.

f. In the Database Home text box, enter the path to the directory where RDBMS server or client software is installed on the RIM host.

g. In the Server ID text box, the information you enter depends on the RDBMS you use with Tivoli Inventory. See the appropriate RDBMS section at the end of this step for the information you need to enter. This information enables the RDBMS to connect to RIM.

h. In the User Name text box, enter the name of the Tivoli Inventory configuration repository user.

i. If you are using DB2, in the Instance Home text box, enter the value of the INSTHOME environment variable. This is the directory where the client instance was created for the DB2 inventory database.

For the installation options specific to your RDBMS, refer to the following sections:

- **DB2**
  - Database ID—Enter the name of the DB2 server database you created for Tivoli Inventory. If you created a remote client that uses an alias, enter the alias name.
  - Database Home—Enter the path to the directory where RDBMS server or client software is installed on the RIM host. For UNIX, this is the value of the environment variable DB2DIR.
  - Server ID—This information enables the RDBMS to connect to RIM. Enter **tcpip**.
  - User Name—Enter the user name for the instance owner authorized to access the Tivoli Inventory database. See “Configuring an RDBMS” on page 2-11 for more information about DB2.
  - Instance Home—Enter the value of the INSTHOME environment variable. This is the directory where the client instance was created for the DB2 Tivoli Inventory database. This text box applies only to DB2 databases.

- **Informix**
  - Database ID—Enter the name of the Informix server database you created for Tivoli Inventory. If you created a remote client that uses an alias, enter the alias name.
  - Database Home—Enter the path to the directory where RDBMS server or client software is installed on the RIM host. For UNIX, this is the value of the environment variable $INFORMIXDIR.
  - Server ID—Specifies the name of the server. This is the value specified by $INFORMIXDIR.
  - User Name—Enter informix.
  - Instance Home—Leave this text box blank; it applies only to DB2.
- **MS SQL Server**
  - Database ID—Enter **inv_db**. This is the default name of the database that is created when you run either the inv_ms_sql_admin.sql or inv_syb_admin.sql script, as described in Chapter 3, "Creating the Configuration Repository."
  - Database Home—Enter the path to the directory where RDBMS server or client software is installed on the RIM host.
  - Server ID—This information enables the RDBMS to connect to RIM. Enter the server name, which is the name of the system on which MS SQL Server is installed.
  - User Name—Enter the name of the user that owns the Tivoli Inventory configuration repository. The default is invtiv.
  - Instance Home—Leave this text box blank; it applies only to DB2.
- **Oracle**
  - Database ID—Enter the value of the ORACLE_SID environment variable. This value is the Oracle instance

ID and is located in the tnsnames.ora file in the $ORACLE_HOME/network/admin directory. The default value that is set during the Oracle installation is ORCL for both Windows NT and UNIX.

- Database Home—Enter the path to the directory where RDBMS server or client software is installed on the RIM host. This is the value of ORACLE_HOME, which is the directory where the Oracle RDBMS software is installed.
- Server ID—This information enables the RDBMS to connect to RIM. Enter the value of the TWO_TASK environment variable. You can find the value of this environment variable in the tnsnames.ora file, which is in the $ORACLE_HOME/network/admin directory.
- User Name—Enter the name of the user that owns the Tivoli Inventory configuration repository. The default is invtiv, if the RIM host is not a Windows NT system.

  If the RIM host is a Windows NT system, enter **invtiv@***TWO_TASK*, where *TWO_TASK* is the value of the TWO_TASK environment variable.
- Instance Home—Leave this text box blank; it applies only to DB2.

■ **Sybase**

- Database ID—Enter **inv_db**. This is the default name of the database that is created when you run either the inv_ms_sql_admin.sql or inv_syb_admin.sql script, as described in Chapter 3, “Creating the Configuration Repository.”
- Database Home—Enter the path to the directory where RDBMS server or client software is installed on the RIM host. This is the value of the environment variable SYBASE, which is the directory where the Sybase RDBMS software is installed. If the RIM host is a different system than the RDBMS server, the value of SYBASE is the directory on the RIM host where the interfaces file is located.

- Server ID—Enter the value of the DSQUERY environment variable. You can find the value of this environment variable in the interfaces file.
- User Name—Enter the name of the user that owns the Tivoli Inventory configuration repository. The default is invtiv.
- Instance Home—Leave this text box blank; it applies only to DB2.

4. Click **Set** to save the values in the Install Options dialog box and return to the Install Product dialog box.
5. In the Clients to Install On scrolling list, select any managed nodes on which you do *not* want to install Tivoli Inventory (only managed nodes are displayed), and click the right arrow button to move the selected managed nodes to the Available Clients scrolling list.
6. To begin installing Tivoli Inventory Server, click **Install**. The Product Install dialog box provides a list of the operations that take place and warns you of any problems that you might want to correct before installing. You can choose one of the following options:
   - Review the status information and click **Continue Install**. The Product Install dialog box informs you when installation is finished.
   - To install Tivoli Inventory Server at another time, click **Cancel**. Your settings are not saved.
7. Update the notice group subscriptions of any administrators who need to receive Tivoli Inventory notices. See the *Tivoli Management Framework User's Guide* for instructions on changing an administrator's notice group subscriptions and other properties.
8. For each policy region in which you plan to create an inventory profile, include the managed resource type InventoryConfig. See the *Tivoli Management Framework User's Guide* for instructions on adding managed resource types to policy regions.

9. Check the Tivoli Inventory notice group to ensure that the RIM object was created. If any RIM creation errors exist, use the **wcrtrim** command to create the RIM object for the RIM host. See the *Tivoli Management Framework Reference Manual* for more information about the **wcrtrim** command.

Changing Passwords for RIM Objects

When you install Tivoli Inventory, the following RIM objects are created:

invdh_1 The inventory data handler uses this RIM object to write scan data to the configuration repository.

inv_query This RIM object is used to perform queries on the configuration repository. The **winvfilter**, **winvrmnode**, and **winvsig** commands and the Tivoli Inventory graphical user interface (GUI) use this RIM object.

In addition, one of the following RIM objects is created:

inv_40 This RIM object is created only if your installation of Tivoli Inventory, Version 4.0, coexists in the same TMR with Tivoli Inventory, Version 3.6.2. This RIM object is used only by Tivoli Software Distribution to access the configuration repository.

inventory This RIM object is created only if your installation of Tivoli Inventory, Version 4.0, does not coexist in the same TMR with Tivoli Inventory, Version 3.6.2. This RIM object is used only by Tivoli Software Distribution to access the configuration repository.

By default, all of the RIM objects are created on the same managed node.

The invdh_1 and inv_query RIM objects both use the Tivoli Inventory, Version 4.0, configuration repository. Software Distribution users can configure the inventory and inv_40 RIM objects to use either the Tivoli Inventory, Version 4.0, configuration repository or the Tivoli Inventory, Version 3.6.2, configuration repository. For more information about the

inv_40 and inventory RIM objects and how to configure them, see the *Tivoli Inventory Release Notes, Version 4.0*.

The default password for all RIM objects is tivoli. However, the password for each RIM object should be the same as the password of the configuration repository that it accesses. Therefore, you must change the password of each RIM object to match that of its configuration repository. In addition, whenever you change the configuration repository password, you must change the passwords for the corresponding RIM objects.

Use the **wsetrimpw** command to change the password for a RIM object. For example, to change the password for the invdh_1, inv_query, and inventory RIM objects, run the following commands:

```
wsetrimpw invdh_1 old_password new_password
wsetrimpw inv_query old_password new_password
wsetrimpw inventory old_password new_password
```

where *old_password* is the current password and *new_password* is the desired new password. See the *Tivoli Management Framework Reference Manual* for more information about the **wsetrimpw** command.

Command Line

The following example uses the **winstall** command to install Tivoli Inventory Server on managed node oak from a CD-ROM image, place the files in the default location, and specify a Sybase RDBMS for the configuration repository. This example applies to all supported RDBMSs except DB2.

```
winstall –c /cdrom -i 40_Inv @INV_DATA_HOST@=oak \
@INV_CB_HOST@=oak @RDBMS_Vendor@=Sybase \
@RDBMS_Host@=olympus @RDBMS_DB_Name@=inv_db \
@RDBMS_DB_Home@=/opt/sybase @RDBMS_DB_Param_one@=INV \
oak @RDBMS_DB_UserName@=invtiv
```

where:

–c /cdrom

Specifies /cdrom as the path to the CD-ROM image.

–i 40_Inv

Specifies the INVENTOR product index file from which Tivoli Inventory is installed.

@INV_DATA_HOST@=oak

Specifies that the managed node oak is the data handler host.

@INV_CB_HOST@=oak

Specifies that the managed node oak is the MDist 2 Callback Host.

@RDBMS_Vendor@=Sybase

Specifies that you are using a Sybase RDBMS with Tivoli Inventory.

@RDBMS_Host@=olympus

Specifies that the system olympus is the RIM host.

@RDBMS_DB_Name@=inv_db

Specifies that the value of the Database ID is inv_db. This is the default name of the database created when you run the inv_syb_admin.sql script.

@RDBMS_DB_Home@=/opt/sybase

Specifies that the value of Database Home is the directory where isql and the interfaces file reside, /opt/sybase.

@RDBMS_DB_Param_one@=INV

Specifies that the value of Server ID is INV. This is the value of the DSQUERY environment variable, which is located in the interfaces file.

oak

Indicates that Tivoli Inventory Server will be installed on the managed node oak. If you do not specify a managed node, the product is installed on all available clients.

@RDBMS_DB_UserName@=invtiv

Specifies that the RDBMS user name for the Tivoli Inventory configuration repository is invtiv.

Note: The preceding command line example can work for every RDBMS that Tivoli Inventory supports except DB2. To use DB2 in the same situation, you need to include the following call in your command:

```
@RDBMS_DB_Param_two@=instance_home
```

where:

@RDBMS_DB_Param_two@=*Instance_Home*

Specifies the directory where the client instance was created for the DB2 Tivoli Inventory database.

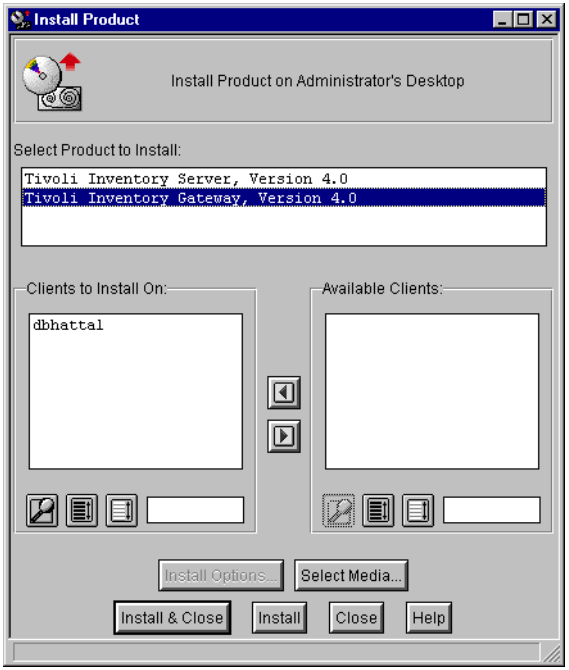
See the *Tivoli Management Framework Reference Manual* for more information about the **winstall** command.

Installing Tivoli Inventory Gateway

Once you have installed Tivoli Inventory Server on your TMR server and selected managed nodes, complete the following steps to install Tivoli Inventory Gateway on managed nodes that you want to act as gateways for managing endpoints:

1. Select **Install –> Install Product** from the Desktop menu. The Install Product dialog box is displayed.

2. In the Install Product dialog box, select **Tivoli Inventory Gateway, Version 4.0** from the Select Product to Install scrolling list.



Choosing **Tivoli Inventory Gateway, Version 4.0** installs files that enable a gateway to perform Tivoli Inventory functions on endpoints. Install this product on all managed nodes that you have configured as gateways.

3. In the Clients to Install On scrolling list, select any managed nodes on which you do *not* want to install Tivoli Inventory Gateway. Only managed nodes are displayed.

4. Click the right arrow button to move the selected managed nodes to the Available Clients scrolling list.

5. To begin installing Tivoli Inventory Gateway, click **Install**. The Product Install dialog box provides a list of the operations that take place and warns you of any problems that you might want to

correct before installing. You can choose one of the following options:

- Review the status information and click **Continue Install**. The Product Install dialog box informs you when installation is finished.
- To install Tivoli Inventory Gateway at another time, click **Cancel**. Your settings are not saved.

Removing Tivoli Inventory

To uninstall Tivoli Inventory, Version 4.0 products from any node in your TMR environment or from the TMR server, you can use either of the following methods:

- An automated uninstall utility that runs product-specific uninstall scripts
- Manual uninstall scripts to selectively remove Tivoli Inventory Server or Tivoli Inventory Gateway

Removing Tivoli Inventory with an Automated Uninstall

Tivoli Management Framework provides a command line utility to remove Tivoli Management Framework applications from a specified node or from the entire TMR. The **wuninst** command is a wrapper script that runs product-specific uninstall scripts.

Using the **wuninst** command, you can remove Tivoli Inventory products from any system in your environment or from the TMR. See the *Tivoli Management Framework Reference Manual* for more information about command line syntax and usage for the **wuninst** command.

To view the usage statement for Tivoli Inventory, enter the following at the command line:

```
wuninst
```

The following example shows how to remove Tivoli Inventory Server from an entire TMR in which the TMR server is named pescado. Note that if the specified node is the TMR server, Tivoli Inventory Server will be removed from the TMR server and from all managed nodes in the

TMR on which Tivoli Inventory Server is installed. Enter the following command:

```
wuninst InventoryServer pescado -rmfiles
```

where:

InventoryServer

Specifies the registered product tag for Tivoli Inventory Server.

pescado

Specifies the TMR server name. Because pescado is the TMR server, Tivoli Inventory Server is removed from every node in the TMR.

–rmfiles

Specifies that all local Tivoli Management Framework database objects and all the associated files are removed regardless of whether they are shared files.

The default behavior without the **–rmfiles** option is to remove the Tivoli Management Framework database objects only for the specified node. If the node is the TMR server, this option removes all database objects.

The following example shows how to remove Tivoli Inventory Gateway from an entire TMR in which the TMR server is named pescado. Enter the following command:

```
wuninst InventoryGateway pescado -rmfiles -all
```

where:

InventoryGateway

Specifies the registered product tag for Tivoli Inventory Gateway.

–rmfiles

Specifies that all local Tivoli database objects and all the associated files are removed regardless of whether they are shared files.

–all

Specifies that Tivoli Inventory Gateway must be removed from all systems in the TMR.

pescado

Specifies the TMR server name. Because pescado is the TMR server, Tivoli Inventory Gateway is removed from every node in the TMR.

The following example shows how to remove database objects from a single node named odin. Enter the following at the command line:

```
wuninst InventoryServer odin
```

where:

InventoryServer

Specifies the registered product tag for Tivoli Inventory Server.

odin

Specifies the node from which the product is removed.

Removing Systems from the Configuration Repository

If you have removed Tivoli Inventory from any systems in your Tivoli environment, you might also find it necessary to remove the information about those systems from the configuration repository. You can do this with the **winvrmnode** command. See Appendix B, "Commands," for more information about this command.

Specifying Tivoli Enterprise Console Servers

If you want to send Tivoli Inventory events to one or more Tivoli Enterprise Console servers that you specify, you must create a Tivoli Inventory rule base and import the BAROC files into the Tivoli Enterprise Console. The Tivoli Inventory application attempts to send events to the Tivoli Enterprise Console servers in the order that they are specified. You do not need to use this procedure if you do not want to send events to a Tivoli Enterprise Console server.

For more information on using Tivoli Inventory with Tivoli Enterprise Console servers, see "wsetinvglobal" on page B-76.

To import the BAROC files into the Tivoli Enterprise Console, perform the following steps:

1. Create a new rule base named Inventory.
2. Copy the default classes from the Tivoli Enterprise Console rules base to the new rule base named Inventory.
3. Import the Tivoli Inventory BAROC files.
4. Compile the Inventory rule base.
5. Load the Inventory rule base into the Tivoli Enterprise Console server.
6. Stop the Tivoli Enterprise Console server.
7. Restart the Tivoli Enterprise Console server.

See the *Tivoli Enterprise Console Rule Builder's Guide* for details on this procedure. The following example creates a new rule base named Inventory and imports the BAROC file for Tivoli Enterprise Console, Version 3.6.*x* or earlier.

Note: The following procedure is not applicable later versions of Tivoli Enterprise Console.

1. Create a new rule base named Inventory.

   ```
   wcrtrb -d hostname:$BINDIR/TME/TEC/rules/Inventory Inventory
   ```

2. Copy the default classes from the Tivoli Enterprise Console rule base to the new rule base named Inventory.

   ```
   wcprb Default Inventory
   ```

3. Import the Tivoli Inventory BAROC files.

   ```
   wimprbclass $BINDIR/TME/INVENTORY/tecad_inv.baroc Inventory
   ```

4. Compile the Inventory rule base.

   ```
   wcomprules Inventory
   ```

5. Load the Inventory rule base into the Tivoli Enterprise Console server.

   ```
   wloadrb Inventory
   ```

6. Stop the Tivoli Enterprise Console server.

   ```
   wstopesvr
   ```

7. Restart the Tivoli Enterprise Console server.

```
wstartesvr
```

Where To Go From Here

Once you have installed Tivoli Inventory products on the systems in your Tivoli environment, go to Chapter 3, “Creating the Configuration Repository,” for information about creating the Tivoli Inventory configuration repository and installing the software signatures and queries provided with Tivoli Inventory, Version 4.0.

3

Creating the Configuration Repository

To store inventory scan results in an RDBMS, you must create the Tivoli Inventory database, or configuration repository. To obtain access to the scan results, you must install the software signatures and queries provided with Tivoli Inventory. This chapter provides the steps necessary to create the Tivoli Inventory configuration repository schema, install software signatures in the configuration repository, and install the queries that you can use to obtain information from the configuration repository.

You must choose a RIM host, install and configure an RDBMS, and install Tivoli Inventory on your TMR before you can create the configuration repository. See Chapter 2, “Installing Tivoli Inventory, Version 4.0,” if you have not done so.

For MS SQL Server, Oracle, and Sybase, Tivoli Inventory includes scripts that create the database and install the configuration repository schema. If you are using DB2 or Informix, you must create the database manually, as described in the section “Configuring DB2” on page 2-12 or “Configuring Informix” on page 2-14. Next, use the scripts provided for DB2 or Informix to create the configuration repository schema.

After you have configured your database, complete the following tasks:

1. Create a database-specific configuration repository using one of the following procedures:
   - "Creating a DB2 Configuration Repository" on page 3-2
   - "Creating an Informix Configuration Repository" on page 3-7
   - "Creating an MS SQL Server Configuration Repository" on page 3-9
   - "Creating an Oracle Configuration Repository" on page 3-13
   - "Creating a Sybase Configuration Repository" on page 3-17
2. Install the software signatures. See "Installing Software Signatures" on page 3-21.
3. Create the Tivoli Inventory and subscription queries. See "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Creating a DB2 Configuration Repository

This section provides the steps required to create a Tivoli Inventory, Version 4.0, configuration repository for a DB2 RDBMS. Before you run any of these scripts, verify that the environment variable DB2INSTANCE is set.

Complete the following tasks before creating your configuration repository:

- If a db2profile script (or the db2cshrc if you are using the C shell) does not reside in the instance owner's home directory that contains the settings for the DB2INSTANCE environment variable, set DB2INSTANCE to the login name of the instance owner.
- Make sure that the path to DB2 is set in the environment from which you are running the scripts.

Creating the User invtiv and the Tivoli Inventory Database

Complete the following steps to run the inv_db2_admin.sql script, which creates the tablespace required for the configuration repository and the temporary tablespace for the instance owner of the Tivoli Inventory configuration repository.

The inv_db2_admin.sql script creates 12,500 pages for a tablespace that is 50 MB. Allow 125 pages or 500 KB for each system whose information will be stored in the database. Thus, if the configuration repository will store information for 1000 nodes, the storage requirement is 125,000 pages or 500 MB. To create larger tablespaces, modify the inv_db2_admin.sql script.

You can change the name and size of the datafile for the tablespace by changing the following line:

```
USING (FILE 'inv_data_1' 12500)
```

to:

```
USING (FILE 'datafile_name' datafile_size)
```

where *datafile_name* is the name that you want to give to the datafile and *datafile_size* is the size that you want the datafile to be.

To create the tablespace required for the Tivoli Inventory configuration repository, complete the following steps:

1. Copy the inv_db2_admin.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory in which you can run DB2 on the DB2 server or the RIM host.
2. Create a connection to the DB2 server database before running the script. Issue the following command at the command line:

   ```
   db2 connect to database_name user user_name using password
   ```

   where:

   database_name

   Specifies the name or alias name of the Tivoli Inventory database.

user_name

Specifies the name of the user who owns the Tivoli Inventory database.This is the login of the DB2 instance owner.

password

Specifies the password of the user referred to by *user_name*.

3. Run the inv_db2_admin.sql script from the command line. Enter the following command:

```
db2 -f inv_db2_admin.sql -o -t -z \
inv_db2_admin.log
```

This command runs the script that creates the tablespaces that the configuration repository uses. This command also directs the output to the screen, specifies a semicolon as the delimiter at the end of a statement, and logs the output in a log file named inv_db2_admin.log. The success or failure of the SQL statements are logged in the log file.

4. Change the password for the instance owner (by default, tivoli) to the password of the instance owner of the configuration repository.

Use the **wsetrimpw** command to change the default password, tivoli, to the password of the instance owner. The instance owner is the user that was specified when you created the DB2 database and instance. The instance owner is also the user name that you enter in the User ID text box on the Install Options dialog box.

For more information about the **wsetrimpw** command, see the *Tivoli Management Framework Reference Manual*.

Installing the Configuration Repository Schema

Complete the following steps to run the inv_db2_schema.sql script, which installs the configuration repository schema. The schema defines the tables and views in the configuration repository. Complete the following steps:

1. Copy the inv_db2_schema.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR

server to a temporary directory on a system with the DB2 client software.

2. Create a connection to the DB2 server database before running the script. Issue the following command at the command line:

```
db2 connect to database_name user user_name \
using password
```

where:

database_name

Specifies the name or alias name of the Tivoli Inventory database.

user_name

Specifies the name of the user who owns the Tivoli Inventory database.

password

Specifies the password of the user referred to by *user_name*.

3. Run the inv_db2_schema.sql script from the command line. Enter the following command:

```
db2 -f inv_db2_schema.sql -o -t \
-z inv_db2_schema.log
```

This command runs the script that installs the configuration repository schema. This command also directs the output to the screen, specifies a semicolon as the delimiter at the end of a statement, and logs the output in a log file named inv_db2_schema.log. The success or failure of the SQL statements are logged in this file.

4. If you are using history tables, repeat steps 1 to 3 with the h_inv_db2_schema.sql script and corresponding h_inv_db2_schema.log log file.

Testing the Configuration

Complete the following steps to test the connection between the RIM host and the RDBMS server and to verify that the configuration repository has been created:

1. On the RIM host, start an interactive SQL session by entering the following command:

   ```
   db2
   ```

2. Test that the client can connect to the Tivoli Inventory database on the DB2 server by entering the following command:

   ```
   connect to aliasname user name using password
   ```

   where:

   aliasname

   Specifies the alias name of the database in the system database directory.

   name

   Specifies the user ID that owns the server database.

   password

   Specifies the password associated with the user specified by *name*.

3. In the DB2 session, test that the configuration repository was installed by entering the following command:

   ```
   select * from INVENTORYDATA
   ```

   Results should indicate that zero rows were found. If results indicate that the view INVENTORYDATA is unknown, the configuration repository was not installed properly. Check the log files for more information.

4. To log out of the DB2 session, enter the following command:

   ```
   quit
   ```

Where to Go from Here

After you have created the DB2 configuration repository, install the software signatures and the Tivoli Inventory and subscription queries. See "Installing Software Signatures" on page 3-21 and "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Creating an Informix Configuration Repository

This section provides the steps required to create the configuration repository with an Informix RDBMS. Before you begin creating an Informix configuration repository, review the following prerequisites:

- RIM for Informix uses the Informix command line interface (CLI), which is Informix's adaptation of ODBC. This release of Tivoli Inventory uses CLI Version 2.5.
- Unlike most Tivoli daemon servers that run as user nobody (or tmersrvd on HP-UX and Windows NT), the daemon for RIM for Informix must run as the user informix. There must be a user named informix on the RIM host.
- The Informix CLI must be installed on the RIM host and the .odbc.ini and .odbcinst.ini files must be installed in the following directories:
  - The root directory
  - The home directory of the informix user
  - The $INFORMIXDIR directory
- Tivoli recommends that you run the database in ANSI mode.
- Follow the installation instructions provided with Informix. Tivoli requires that you install Informix in the following order for UNIX:
  1. Informix's Embedded SQL (ESQL) products
  2. Informix CONNECT
  3. Informix CLI 2.5
  4. Informix server

For more information, see your Informix documentation.

Creating the Informix User and the Tivoli Inventory Database

Refer to your Informix documentation for information about creating the database and the database user. When determining the size of the database, allow 1 megabyte (MB) for each machine you intend to scan.

Use the **wsetrimpw** command to change the default password, tivoli, to the password of the instance owner. The instance owner is the user that was specified when you created the Informix database and instance. The instance owner is also the user name that you enter in the User ID text box on the Install Options dialog box.

For more information about the **wsetrimpw** command, see the *Tivoli Management Framework Reference Manual*.

Installing the Configuration Repository Schema

After you have installed Tivoli Inventory, create a Tivoli Inventory configuration repository in the Informix RBDMS by completing the following steps:

1. Verify that you have set at least 20,000 locks in the onconfig file before you run the script to install the schema. This is also the minimum number of locks necessary for Tivoli Inventory.
2. Copy the inv_infx_schema.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory to a temporary directory in which you can run Informix on the Informix server.
3. From the directory that now contains the script, run the inv_infx_schema.sql script. Enter the following command:

   ```
   dbaccess db_name@$INFORMIXSERVER inv_infx_schema.sql >> \
   inv_infx_schema.log 2>&1
   ```

   where *db_name* is the name of the Tivoli Inventory configuration repository.
4. If you are using history tables, repeat steps 1 to 3 with the h_inv_infx_schema.sql script.

Testing the Configuration

Complete the following steps to test the connection between the RIM host and the RDBMS server and to verify that the configuration repository has been created:

1. On the RIM host, start an interactive SQL session by entering the following command:

   ```
   dbaccess db_name@$INFORMIXSERVER
   ```

   where *db_name* is the name of the database and *INFORMIXSERVER* is the name of the Informix server.

2. In the Informix session, select Query-Language.
3. Select New.
4. Enter the following command:

   ```
   select * from INVENTORYDATA
   ```

5. Select Run to test that the configuration repository was installed.

   Results should indicate that zero rows were found. If results indicate that the view INVENTORYDATA is unknown, the configuration repository was not installed properly. Check the log files for more information.

6. Select Exit to log out of the Informix session.

Where to Go from Here

After you have created the Informix configuration repository, install the software signatures and the Tivoli Inventory and subscription queries. See "Installing Software Signatures" on page 3-21 and "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Creating an MS SQL Server Configuration Repository

This section provides the steps required to create the configuration repository with an MS SQL Server RDBMS. Before creating your configuration repository, ensure that the path to isql is set in the environment from which you are running the scripts.

Creating the User invtiv and the Tivoli Inventory Database

You must use the inv_ms_sql_admin.sql script as a template to create a database, the user invtiv, and the password tivoli in the MS SQL Server RDBMS.

By default, the script creates a database that is 20 MB, which stores information for about 40 PCs. Allow 500 KB for each system whose information will be stored in the database. Thus, if the configuration repository will store information for 1000 nodes, the storage requirement is 500 MB.

As provided, the inv_ms_sql_admin.sql script uses the master *device* to create the configuration repository. For MS SQL Server, the term device refers to the space, allocated on a server, on which a user can create a database. The master device must be dedicated to the master database and system functions only. Therefore, before you run the script, you must change it by completing the following steps:

1. Create a new device. To create a new device, complete the following steps:
   a. Determine the size of the new device for your Tivoli Inventory configuration repository. It must be at least as large as the database you create when you run the inv_ms_sql_admin.sql script.
   b. Create the new device. See your MS SQL Server documentation for more information about creating new devices.
2. Copy the inv_ms_sql_admin.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory on a system with the MS SQL client software installed.
3. Edit the script to use the device you created in step 1.

   To modify the script, change the following line:

   ```
   create database inv_db on master = 20
   ```

   to:

   ```
   create database inv_db on new_device = size
   ```

where:

new_device

Specifies the name of the device you created in step 1.

size

Specifies the size of the database you create when you run the script. The size of the database cannot be larger than the size of *new_device*.

After you have edited the script to specify the correct device and size, complete the following steps to create the configuration repository:

1. From the directory that now contains the script, run the inv_ms_sql_admin.sql script as user sa. Enter the following command:

   ```
   isql -U sa [-P password] -i inv_ms_sql_admin.sql \
   -o inv_ms_sql_admin.log
   ```

   where *password* is the RDBMS password that is set for the RDBMS user sa.

   The script creates the user invtiv (with the password tivoli) and inv_db database in the MS SQL Server RDBMS. The script also logs the output to the inv_ms_sql_admin.log file. The success or failure of the SQL statements are logged in this file.

2. For security reasons, Tivoli recommends that you change the password for the user invtiv. Use the appropriate isql command to change the password, and then use the **wsetrimpw** command to notify the Tivoli Management Framework and Tivoli Inventory of the password change.

   For more information about the **wsetrimpw** command, see the *Tivoli Management Framework Reference Manual*.

Installing the Configuration Repository Schema

Complete the following steps to run the inv_ms_sql_schema.sql script, which installs the configuration repository schema. The schema defines

the tables, and views in the configuration repository. You must run the script as the RDBMS user invtiv. Complete the following steps:

1. Copy the inv_ms_sql_schema.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory in which you can run isql on a system with the MS SQL client software installed.

2. From the directory that now contains the script, run the inv_ms_sql_schema.sql script as user invtiv. Enter the following command:

   ```
   isql -U invtiv [-P password] \
   -i inv_ms_sql_schema.sql \
   -o inv_ms_sql_schema.log
   ```

   where *password* is the RDBMS password that is set for the MS SQL Server user invtiv.

   The script installs the configuration repository tables and views and logs the output to the inv_ms_sql_schema.log file. The success or failure of the SQL statements is logged in this file.

3. If you are using history tables, repeat steps 1 to 2 with the h_inv_ms_sql_schema.sql script and corresponding h_inv_ms_sql_schema.log log file.

Testing the Configuration

Complete the following steps to test the connection between the RIM host and the RDBMS server and to verify that the configuration repository has been created:

1. From the RIM host, test that the MS SQL Server user invtiv is a valid user and that the client can connect to the RDBMS by entering the following command:

   ```
   isql -U invtiv -P password
   ```

   where *password* is the RDBMS password that is set for the MS SQL Server user invtiv.

2. In the isql session, test that the configuration repository was installed by entering the following command:

   ```
   select * from INVENTORYDATA
   go
   ```

Results should indicate that zero rows were found. If results indicate that the view INVENTORYDATA is unknown, the configuration repository was not installed properly. Check the log files for more information.

3. To log out of the isql session, enter the following command:

```
exit
```

Where to Go from Here

After you have created the MS SQL Server configuration repository, install the software signatures and the Tivoli Inventory and subscription queries. See "Installing Software Signatures" on page 3-21 and "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Creating an Oracle Configuration Repository

This section provides the steps required to create the configuration repository with an Oracle RDBMS. Check the following items in your environment before creating your configuration repository:

- Set the environment variable ORACLE_HOME to be the path to the directory where the Oracle RDBMS client or server software is installed.
- Also, if the RIM host is a different machine from the RDBMS server, you must set the environment variable TWO_TASK. TWO_TASK is the identifier that the client uses to extract information from the tnsnames.ora file.
- Ensure that the path to sqlplus is set in the environment from which you are running the scripts.

Creating the User invtiv and the Tivoli Inventory Database

Complete the following steps to run the inv_ora_admin.sql script, which creates the tablespace required for the configuration repository, the temporary tablespace for the user invtiv, with the password tivoli in the Oracle RDBMS. The script creates a tablespace that is 50 MB. Allow 500 KB for each system whose information will be stored in the

database. Thus, if the configuration repository will store information for 1000 nodes, the storage requirement is 500 MB. To create larger tablespaces, modify the inv_ora_admin.sql script.

You can make the following changes to the inv_ora_admin.sql script:

- To change the size of the data tablespace, change the following line:

  ```
  datafile 'inv_data_1' size 50M
  ```

  to:

  ```
  datafile 'inv_data_1' size tablespace_size
  ```

  where *tablespace_size* is the size that you want the tablespace to be.

- To change the size of the temporary tablespace, change the following line:

  ```
  datafile 'inv_temp_1' size 10M
  ```

  to:

  ```
  datafile 'inv_temp_1' size temp_tablespace_size
  ```

  where *temp_tablespace_size* is the size that you want the temporary tablespace to be.

To create the tablespaces required for the Tivoli Inventory configuration repository, complete the following steps:

1. Copy the inv_ora_admin.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory on a system with the Oracle client software installed.
2. From the directory that now contains the script, start an SQL*Plus session and log in to the RDBMS as the user system. Enter the following command:

   ```
   sqlplus system/password
   ```

   where *password* is the RDBMS password that is set for the RDBMS user system.
3. To capture the log information into a file, enter the following command:

   ```
   spool inv_ora_admin.log
   ```

4. In the SQL*Plus session, run the inv_ora_admin.sql script. Enter the following command:

   ```
   @inv_ora_admin.sql
   ```

   The script creates the user invtiv (with the password tivoli) and the necessary tablespaces for the configuration repository. The success or failure of the SQL statements is logged in the log file.

5. To log out of SQL*Plus, when the script is finished, enter the following command:

   ```
   quit
   ```

6. For security reasons, Tivoli recommends that you change the password for the user invtiv. Use the appropriate SQL*Plus command to change the password, and then use the **wsetrimpw** command to notify the Tivoli Management Framework and Tivoli Inventory of this password change.

   For more information about the **wsetrimpw** command, see the *Tivoli Management Framework Reference Manual*.

Installing the Configuration Repository Schema

Complete the following steps to run the inv_ora_schema.sql script, which installs the configuration repository schema. The schema defines the tables and views in the configuration repository. You must run the script as the RDBMS user invtiv. Complete the following steps:

1. Copy the inv_ora_schema.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory on a system with the Oracle client software installed.

2. From the directory that now contains the script, start an SQL*Plus session and log in to the RDBMS server as the RDBMS user invtiv. Enter the following command:

   ```
   sqlplus invtiv/password
   ```

   where *password* is the RDBMS password that is set for the Oracle user invtiv.

3. To capture the log information into a file, enter the following command:

   ```
   spool inv_ora_schema.log
   ```

4. In the SQL*Plus session, run the inv_ora_schema.sql script. Enter the following command:

   ```
   @inv_ora_schema.sql
   ```

   The script installs the configuration repository tables and views. The success for failure of the SQL statements is logged in the log file.

5. To log out of SQL*Plus, when the script is finished, enter the following command:

   ```
   quit
   ```

6. If you are using history tables, repeat steps 1 to 5 with the h_inv_ora_schema.sql script and corresponding h_inv_ora_schema.log log file.

Testing the Configuration

Complete the following steps to test the connection between the RIM host and the RDBMS server and to verify that the configuration repository has been created:

1. From the RIM host, test that the Oracle user invtiv is a valid user and that the RIM host can connect to the Tivoli Inventory database on the Oracle server by entering the following command:

   ```
   sqlplus invtiv/password
   ```

   where *password* is the RDBMS password that is set for the Oracle user invtiv.

2. In the SQL*Plus session, test that the configuration repository was installed in the Oracle RDBMS by entering the following command:

   ```
   select * from INVENTORYDATA;
   ```

   Results should indicate that zero rows were found. If results indicate that the view INVENTORYDATA is unknown, the configuration repository was not installed properly. Check the log files for more information.

3. To log out of SQL*Plus, enter the following command:

```
quit
```

Where to Go from Here

After you have created the Oracle configuration repository, install the software signatures and the Tivoli Inventory and subscription queries. See "Installing Software Signatures" on page 3-21 and "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Creating a Sybase Configuration Repository

This section provides the steps required to create the configuration repository with a Sybase RDBMS. Check the following items before creating your configuration repository:

- Set the SYBASE environment variable as the path to the directory where the Sybase RDBMS client or server software is installed.
- Set the DSQUERY environment variable as the identifier that the client uses to extract connection information from the interfaces file.
- Ensure that the path to isql is set in the environment from which you are running the scripts.

Creating the User invtiv and the Tivoli Inventory Database

You must use the inv_syb_admin.sql script as a template to create a database and the user invtiv with the password tivoli in the Sybase RDBMS.

By default, the script creates a database that is 20 MB, which stores information for about 40 PCs. Allow 500 KB for each system whose information will be stored in the database. Thus, if the configuration repository will store information for 1000 nodes, the storage requirement is 500 MB.

As provided, the inv_syb_admin.sql script uses the master device to create the configuration repository. For Sybase, the term *device* refers to the space, allocated on a server, with which a user can create a database.

The master device must be dedicated to the master database and the system functions only. Therefore, before you run the script, you must change it by completing the following steps:

1. Create a new device. To create a new device, complete the following steps:
   a. Determine the size of the new device for your Sybase configuration repository. It must be at least as large as the database you create when you run the script in step 4.
   b. Create the new device. See your Sybase documentation for more information about creating new devices.
2. Copy the inv_syb_admin.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory on a system with the Sybase client software installed.
3. Edit the script so that it uses the device that you created in step 1. To modify the script, change the following line:

   ```
   create database inv_db on master = 20
   ```

   to:

   ```
   create database inv_db on new_device = size
   ```

   where:

   new_device

   Specifies the name of the device you create in step 1.

   size

   Specifies the size of the database you create when you run the script. The size of the database cannot be larger than the size of *new_device*.

After you have edited the script to specify the correct device and size, complete the following steps to create the configuration repository:

1. From the directory that now contains the script, run the inv_syb_admin.sql script as the user sa. Enter the following command:

   ```
   isql -U sa -P password -i inv_syb_admin.sql \
   -o inv_syb_admin.log
   ```

   where *password* is the RDBMS password that is set for the RDBMS user sa.

   The script creates the user invtiv (with the password tivoli) and inv_db database in the Sybase RDBMS and logs the output to the inv_syb_admin.log file.

2. For security reasons, Tivoli recommends that you change the password for the user invtiv. Use the appropriate isql command to change the password, and then use the **wsetrimpw** command to notify the Tivoli Management Framework and Tivoli Inventory of the password change.

   For more information about the **wsetrimpw** command, see the *Tivoli Management Framework Reference Manual*.

Installing the Configuration Repository Schema

Complete the following steps to run the inv_syb_schema.sql script, which installs the configuration repository schema. The schema defines the tables, and views in the configuration repository. You must run the script as the RDBMS user invtiv. Complete the following steps:

1. Copy the inv_syb_schema.sql script from the $BINDIR/../generic/inv/SCRIPTS/RDBMS directory on the TMR server to a temporary directory in which you can run isql on a system with the Sybase client software installed.

2. From the directory that now contains the script, run the inv_syb_schema.sql script as the user invtiv. Enter the following command:

   ```
   isql -U invtiv -P password -i inv_syb_schema.sql \
   -o inv_syb_schema.log
   ```

   where *password* is the RDBMS password that is set for the Sybase user invtiv.

   The script installs the configuration repository tables and views and logs the output to the inv_syb_schema.log file.

3. If you are using history tables repeat steps 1 to 2 with the h_inv_syb_schema.sql script and corresponding h_inv_syb_schema.log log file.

Testing the Configuration

Complete the following steps to test the connection between the RIM host and the RDBMS server and to verify that the configuration repository has been created:

1. From the RIM host, test that the Sybase user invtiv is a valid user and that the RIM host can connect to the Sybase RDBMS by entering the following command:

   ```
   isql -U invtiv -P password
   ```

   where *password* is the RDBMS password that is set for the Sybase user invtiv.

2. In the isql session, test that configuration repository was installed in the Sybase RDBMS by entering the following command:

   ```
   select * from INVENTORYDATA
   go
   ```

   Results should indicate that zero rows were found. If results indicate that the view INVENTORYDATA is unknown, the configuration repository was not installed properly. Check the log files for more information.

3. To log out of isql, enter the following command:

   ```
   quit
   ```

Where to Go from Here

After you have created the Sybase configuration repository, install the software signatures and the Tivoli Inventory and subscription queries. See "Installing Software Signatures" on page 3-21 and "Installing Tivoli Inventory and Subscription Queries" on page 3-22.

Installing Software Signatures

As part of the Tivoli Inventory installation process, you must install a set of software signatures in the configuration repository. A software signature is the set of information that identifies a certain software package, such as the name and size of the executable file for the software package. Tivoli Inventory uses software signatures to determine which software packages are installed on the machines you scan. When you run a software signature scan on an endpoint, Tivoli Inventory distributes the software signatures to the endpoint, and then compares each file on the endpoint to the list of software signatures. When a file matches a software signature, the software signature data for that file is sent to the configuration repository.

The default set of software signatures that are provided with Tivoli Inventory are in a file named SWSIGS.INI, which is installed on the TMR server and managed nodes when you install Tivoli Inventory. You must use the **winvsig** command to install the software signatures that are in the SWSIGS.INI file into the configuration repository.

The following table provides the authorization roles required to install or update the Tivoli Inventory software signatures:

Operation	Context	Role
Install Tivoli Inventory software signatures	Policy region	**RIM_update** or **super**

To install the Tivoli Inventory software signatures in the configuration repository, use the **–f** option of the **winvsig** command to add the

software signatures defined in the SWSIGS.INI file. Enter the following command:

```
winvsig -a -f $BINDIR/../generic/inv/SIGNATURES/SWSIGS.INI
```

where:

–a

Indicates that software signatures are to be added to the configuration repository.

–f

Specifies that the software signatures to be added are contained in a file.

$BINDIR/**../generic/…**

Specifies the directory where the **SWSIGS.INI** file is located.

SWSIGS.INI

Specifies the name of the file that contains the software signatures to be installed in the configuration repository. To install all the software signatures, use the SWSIGS.INI file.

See Chapter 7, "Collecting Custom Information with Tivoli Inventory," for more information about software signatures. See Appendix B, "Commands," for more information on the **winvsig** command.

Installing Tivoli Inventory and Subscription Queries

To access the information gathered by an Tivoli Inventory hardware or software scan, use the Tivoli Management Framework query facility. Tivoli Inventory provides four sets of pre-defined queries—the inventory queries, the subscription queries, the historical inventory queries, and the historical subscription queries. You install these queries when you run the inventory_query.sh, subscription_query.sh, h_inventory_query.sh, and the h_subscription_query.sh scripts, all of which are provided with Tivoli Inventory. Each script creates a query

library that contains a set of queries designed to access the information that Tivoli Inventory stores in the configuration repository.

- The inventory_query.sh script creates the INVENTORY_QUERY query library that contains queries that return general inventory information about the machines in your TMR.
- The subscription_query.sh script creates the SUBSCRIPTION_QUERY query library that contains queries that return lists of possible subscribers for profiles based on various scan results.
- The h_inventory_query.sh script creates the H_INVENTORY_QUERY query library that contains history queries that return general historical inventory information about the machines in your TMR.
- The h_subscription_query.sh script creates the H_SUBSCRIPTION_QUERY query library that contains history queries that return lists of possible subscribers for profiles based on various scan results.

The scripts can be found in the *$BINDIR*/../generic/inv/SCRIPTS/QUERIES directory. When you run the scripts, you can rename either of the query libraries. Run the scripts on the TMR server for the TMR on which you want to use the queries. Because queries and query libraries must have unique names in a TMR, you can have only one set of the Tivoli Inventory queries in a TMR or in a set of interconnected TMRs.

See Appendix C, "Configuration Repository," for a description of the queries that these scripts create. See Chapter 8, "Querying Inventory Information," for information about creating new queries to access Tivoli Inventory information.

The following table provides the authorization roles required to create the Tivoli Inventory queries:

Operation	Context	Role
Create inventory and subscription queries	Policy region	**admin**, **senior**, or **super**

Complete the following steps to run the inventory_query.sh, subscription_query.sh, h_inventory_query.sh, or h_subscription_query.sh script:

1. Select the policy region in which you want to create the query libraries.
2. Add the QueryLibrary managed resource type to the policy region. See the *Tivoli Management Framework Reference Manual* for more information about adding managed resources to a policy region.
3. Verify that your Tivoli environment is set. If it is not, run the /etc/Tivoli/setup_env.sh script.
4. Run each script by entering the following command:

```
$BINDIR/../generic/inv/SCRIPTS/QUERIES/scriptname
region_name query_library_name
```

where:

scriptname

Specifies inventory_query.sh, subscription_query.sh, h_inventory_query.sh, or h_subscription_query.sh as the name of the script.

region_name

Specifies the name of the policy region in which the query library is created.

query_library_name

(Optional) Specifies a different name for the query library than the default included with the script.

After you have run the scripts, you have four new query library icons in your policy region window in the Tivoli Desktop as shown in the following example:

Where to Go from Here

After you have created the configuration repository, installed the software signatures, and installed the queries, you should configure Scalable Collection Service.

4

Working with Scalable Collection Service

After you install Tivoli Inventory and create the configuration repository, it is recommended that you configure your Tivoli environment to use Scalable Collection Service (SCS). SCS uses a hierarchy of collectors to transfer data from endpoints to the inventory data handler. The data is then passed through one or more RIM objects to the configuration repository. You should optimize the SCS components to produce an efficient flow of data and prevent errors during data transfer. This chapter describes how Tivoli Inventory and SCS work together and provides information about configuring and using Tivoli Inventory with SCS.

This chapter includes the following:

- An explanation of how Tivoli Inventory and SCS gather scan data and store it in the configuration repository. (See "How SCS Works with Tivoli Inventory" on page 4-2.)
- Information about stopping, starting, resetting, and configuring collectors. (See "Using Collectors" on page 4-4.)
- Information about configuring the inventory data handler to write data to the RIM object and manage scan status information, and the procedure to move the inventory data handler. (See "Using the Inventory Data Handler" on page 4-7.)
- The procedure to move the inventory callback object. (See "Moving the Inventory Callback Object" on page 4-10.)

- Information about RIM objects. (See “Creating and Configuring RIM Objects for the Inventory Data Handler” on page 4-11.)
- The procedure to schedule the transfer of collected scan data through the collection hierarchy. (See “Scheduling Collections” on page 4-14.)
- An explanation of how you can control the way SCS sends data through your network. (See “Controlling the Flow of Network Data” on page 4-17.)
- Procedures you can use to view information about collectors, collection tables of contents (CTOCs), queues, the inventory data handler, scans, and inventory profiles, as well as information about viewing data in the configuration repository. (See “Obtaining Information about SCS” on page 4-18.)
- An explanation of how you can configure Tivoli Inventory to gather and return status information about scans. (See “Configuring Tivoli Inventory Status Information” on page 4-21.)
- Information about using SCS across Tivoli Management Regions (TMRs). (See “Considerations for Interconnected TMRs” on page 4-22.)

How SCS Works with Tivoli Inventory

The following table describes how Tivoli Inventory and SCS interact to store inventory data in the configuration repository:

Action	Product or Service
You create an inventory profile.	Tivoli Inventory
You distribute the profile to endpoints.	Tivoli Inventory
The profile distribution completes.	Tivoli Inventory

Action	Product or Service
The following events occur on an endpoint as the scan of the endpoint completes: 1. A MIF file is created. 2. The MIF file is parsed. 3. Scan data is generated. 4. The scan data is encoded, compressed, and saved in a file.	Tivoli Inventory
After the scan finishes on an endpoint, the endpoint makes an upcall to the gateway to request data collection by sending a CTOC. **Note:** If the endpoint cannot successfully forward the CTOC to the collector, the scan data for that endpoint is sent without using SCS.	SCS
The gateway forwards the CTOC to the local collector.	SCS
When the collector on the gateway receives the CTOC, the following events occur: 1. The collector places the CTOC in the input queue. 2. The collector makes a downcall to the endpoint to retrieve the compressed and encoded data file. 3. The collector stores the data file in the collector depot. The CTOC moves from the input queue to the output queue. 4. The collector requests data collection from the next collector in the hierarchy (the upstream collector) by sending the CTOC to the upstream collector. 5. The upstream collector retrieves the data from the downstream collector. When the data transfer is complete, the CTOC in the output queue of the downstream collector is discarded. 6. Steps 1 through 5 repeat until the data file reaches the inventory data handler.	SCS

Action	Product or Service
The inventory data handler stores CTOCs in an input queue and scan data in a depot in the same way that collectors do. However, rather than sending a collection request to another collector, the inventory data handler uncompresses the scan data and sends it to an available RIM object.	Tivoli Inventory
From the RIM object or objects, the information is sent to the configuration repository. The status collector collects the final status for each endpoint.	Tivoli Inventory

To modify an SCS hierarchy, you must use the **wrpt** command to set up your repeater topology as needed, and then use the **wcollect** command with the **–r** option on the collectors in the modified part of the hierarchy to reset the collector routing topology. (See “Using Collectors” on page 4-4.) Because SCS shares the multiplexed distribution (MDist 2) repeater hierarchy (although data flows in the opposite direction), you must configure the hierarchy so that it is optimized for both services. See the *Tivoli Management Framework Reference Manual* for more information about the **wrpt** command.

Using Collectors

This section discusses the various tasks you can perform on collectors using the **wcollect** command.

Note: When using the **wcollect** command, you must specify the collector on which you want to perform the action. The collector name is always the name of the managed node or gateway on which the collector is installed.

For more information about the **wcollect** command, see “wcollect” on page B-8.

Stopping, Starting, and Resetting a Collector

You must stop and restart a collector to perform certain tasks with SCS. For example, you must stop and restart a collector after you reconfigure

it for the changes to take effect. Also, you must stop a collector if you create a new run-time directory, so that you can move data from the old directory to the new directory. You may also need to stop and restart a collector during troubleshooting.

SCS provides two options for stopping a collector using the **wcollect –h** option:

- **graceful**—Bringing the collector to a graceful halt stops the collector after it completes all active collections.
- **immediate**—Bringing the collector to an immediate halt stops the collector without waiting for active collections to finish processing. When you restart the collector, any collections that were active when the collector was halted are automatically restarted, so no data is lost.

Once you have stopped a collector using the **–h** option, you must use the **–s** option to start the collector again.

If you have made changes to the collector hierarchy using the **wrpt** command, you must reset the collector topology for each affected collector. Use the **–r** option of the **wcollect** command to remove the locally cached collection route for a collector. After you execute this command, the collector downloads new routing information from the collection manager when the collector processes the next collection request.

Configuring a Collector

After you install Tivoli Inventory, you should configure collectors to ensure that the run-time directory and depot are the correct size, logging options are set to your needs, and threads are set to optimize communication between collectors. You might also need to reconfigure collectors after making changes to the collector hierarchy. The configuration options for collectors are described in the following sections.

Configuring the Run-time Directory and Depot

The run-time directory contains the depot and run-time (*.dat and *.log) files. By default, the SCS run-time directory is at $DBDIR/mcollect on

each collector. The depot is located within the run-time directory at $DBDIR/mcollect/depot.

You can move the run-time directory and depot if space is limited in the $DBDIR directory. For example, you might choose to move the depot to a larger file system if you scan very large numbers of endpoints or if the depot shares disk space with the oserv, which also writes data to the $DBDIR directory.

To move the run-time directory and depot, first specify a new location for the run-time directory using the **wcollect –l** option. Next, manually move the depot and all other data from the old run-time directory to the new directory. The run-time directory must reside on a stable disk with a large amount of free space to ensure persistent storage of collections. This directory should not be a temporary directory. You must also provide read-write privileges for the tmersrvd or nobody account (the Tivoli unprivileged account) in this directory.

Note: After you specify a new run-time directory, you must stop the collector before moving the depot and data from the old directory to the new directory.

You can change the size of the depot using the **wcollect –z** option. By default, the depot size is 40 megabytes (MB), but you can make it as large as necessary. Note that you cannot make it smaller; you can only make it larger. For example, you can set the depot to 50 MB and later change that to 55 MB, but you cannot change the size to 45 MB.

You can control the size of data units that are transferred between collectors. These units of data are referred to as *transmission chunks*. You set the transmission chunk size with the **wcollect –c** option. The default size is 1 MB.

If a collector attempts to collect scan data but the depot is too small to hold the data, the collection for the scanned endpoint fails. For more information, see “Troubleshooting Procedures” on page E-13.

Configuring the Collector Log File

The mcollect.log file contains all the activity of a collector as data flows through it. You control the amount of information that is logged in this file by setting a value with the **wcollect –d** option. By default, only fatal errors are logged. You specify the maximum size of this file using the **wcollect –g** option. By default, the maximum size is 1 MB.

Configuring Threads

Threads control each successful communication attempt between two collectors. Collectors run as multithreaded daemon processes. The collector uses threads primarily to maintain multiple data streams going into and out of the depots. Several configuration options of the **wcollect** command are related to threads between collectors. You can set the maximum number of input threads with the **–t** option and output threads with the **–o** option. The default number for both types of threads is 5.

An input thread becomes idle when no requests in its queues are ready to process. You can specify the amount of time that an input thread should wait before it shuts down by using the **–i** option. The default idle time is 60 seconds. To specify the amount of time that a thread waits if system or network resources are temporarily unavailable, use the **–p** option. The default value is 5 seconds.

Retries are related to threads. A *retry* is an attempt by a collector to process a collection request from another collector following a failure. Data transfer between collectors, or between collectors and endpoints, can fail for a variety of reasons. For example, if a thread cannot negotiate a network connection to another collector or endpoint, the thread has failed. By configuring retries and a retry delay time for collector threads, you enable SCS to recover from temporary error conditions.

Use the **wcollect –m** option to set the maximum number of times that a collector attempts to process a collection request from a downstream collector following a failure. By default, this value is 10.

Use the **wcollect –e** option to set the retry delay time. This is the time in seconds that a collector waits before trying again to establish communication with another collector. The default value is 1 second.

Using the Inventory Data Handler

The inventory data handler can be considered the last collector in the SCS collector hierarchy. The inventory data handler receives data from collectors and sends the data to the configuration repository.

The inventory data handler is created automatically on the managed node that you specify during installation. If you do not specify a managed node, the inventory data handler is created on the TMR server.

However, it is recommended that you do *not* install the inventory data handler on the same managed node as the TMR server.

This section describes how to configure the inventory data handler and how to move it to a different managed node.

Configuring the Inventory Data Handler

The inventory data handler is created when you install Tivoli Inventory. If you need to move the inventory data handler to a different managed node, you can do so using the **wcrtinvdh** command. You use the **wsetinvdh** command to specify how the inventory data handler writes data to a RIM object. You can also use this command to configure status collector options, which specify how notifications of scan status are collected, stored, and distributed. You use the **wcollect** command to configure output threads for the inventory data handler.

This section describes the options you can use to configure the inventory data handler. For more information about the **wcrtinvdh**, **wsetinvdh** and **wcollect** commands, see Appendix B, "Commands."

You use the **wsetinvdh –s** option to specify whether the inventory data handler stores status information. *Status information* is a record of whether the scan has completed on each endpoint in a profile distribution. If a system failure occurs, status information is automatically restored when the inventory data handler is back online. By default, status information is stored.

You use the **wsetinvdh –d** option to specify the directory where the inventory data handler stores status information. The default location for this directory is $DBDIR/inventory/stat_dir on the managed node where the inventory data handler resides. You can specify a different directory.

Using the **wsetinvdh –n** option, you can specify the interval, in minutes, at which status information is sent. For example, if you set this value to 10, every 10 minutes the inventory data handler sends all completed scan notifications in a single notice called a *bundle*. The inventory data handler sends the bundles to the Tivoli Inventory notice group, a log file, the TEC console, or a combination of those options, depending on the setting of the **wsetinvglobal –l** option. The default value for the **wsetinvdh –n** option is 10 minutes.

Using the **wsetinvdh –q** option, you can specify the maximum number of targets in a bundle. For example, if you set this value to 5, each time scans complete on five targets, the inventory data handler bundles the status information and sends it. The default value is 10 targets.

The **wsetinvdh –n** and **–q** options, in combination with the **–n** option of the **wsetinvglobal** command, configure status information to be sent in bundles.

Notes

- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.
- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.
- Bundling occurs only when the **–n** option for the **wsetinvglobal** command is set to BUNDLE.

You use the **wcollect –o** option to specify the number of output threads for the inventory data handler to write to RIM objects. It is recommended that this value match the total number of RDBMS connections set for all RIM objects used by the inventory data handler. For example, if the inventory data handler uses two RIM objects to connect to the RDBMS, and each RIM object has 5 RDBMS connections, set the output threads for the inventory data handler to 10. For information about setting RDBMS connections for RIM objects, see "Configuring RIM Objects" on page 4-13.

The **wsetinvdh –r** option enables you to specify the number of times the inventory data handler tries to write data to a RIM object. After the specified number of retries is reached, the inventory data handler sends a failure notification. The default value for this option is 5 retries.

Related to the **wsetinvdh –r** option, the **wsetinvdh –t** option enables you to specify a value from which to calculate the timeout period between retries. The inventory data handler calculates the timeout period using the algorithm *timeout*retry_count*. For example, if you set this value to 20 seconds, on the first retry the algorithm sets the timer to

20 * 1 or 20 seconds. On the second retry, the timer sets to 20 * 2 or 40 seconds. The default value for the **wsetinvdh –t** option is 30 seconds.

Moving the Inventory Data Handler

To move the inventory data handler, delete it from the managed node where it currently resides, and then create a new inventory data handler on a different managed node.

Note: Before moving the inventory data handler, make sure that no inventory scans are active. If you move the inventory data handler before scans complete, you will lose any data that has not reached the configuration repository.

Perform the following tasks to move the inventory data handler:

1. Make sure that the SCS patch and the Tivoli Inventory product have been installed on the managed node where you will create the new inventory data handler.
2. Delete the existing inventory data handler by running the following command:

   ```
   wdel @InvDataHandler:inv_data_handler
   ```

   For more information about the **wdel** command, see the *Tivoli Management Framework Reference Manual*.
3. Create the new inventory data handler using the **wcrtinvdh** command and appropriate options. For more information, see “wcrtinvdh” on page B-18.

Moving the Inventory Callback Object

The inventory callback object performs the following functions:

- When inventory data cannot be collected using the collector hierarchy, MDist 2 sends the data to the inventory callback object, which sends it to the inventory data handler. The data is then sent through one or more RIM objects to the configuration repository.
- For all scans, MDist 2 sends a status message to the inventory callback object indicating whether it successfully delivered the inventory profile to the endpoint. This message indicates only that

the endpoint processed the profile, not that the scan data successfully reached the configuration repository.

When you install Tivoli Inventory, the inventory callback object is created automatically on the managed node that you specify. To move the inventory callback object, you delete it from the managed node where it currently resides, and then create a new inventory callback object on a different managed node. You can create the inventory callback object on any managed node in your TMR.

To move the inventory callback object, you delete it from the managed node where it currently resides, and then create a new inventory callback object on a different managed node.

Note: Before moving the inventory callback object, make sure that no inventory scans are active.

Perform the following tasks to move the inventory callback object:

1. Make sure that the SCS patch and the Tivoli Inventory product have been installed on the managed node where you will create the new inventory callback object.
2. Delete the existing inventory callback object by running the following command:

   ```
   wdel @InventoryConfigCB:inv_cb
   ```

   For more information about the **wdel** command, see the *Tivoli Management Framework Reference Manual*.
3. Create the new inventory callback object using the **wcrtinvcb** command and appropriate options. For more information about this command, see “wcrtinvcb” on page B-16.

Creating and Configuring RIM Objects for the Inventory Data Handler

The inventory data handler uses a RIM object to write data to the configuration repository. When you install Tivoli Inventory, the invdh_1 RIM object is created for this purpose. However, you can create one or more additional RIM objects for the inventory data handler to use. This section describes how to create these additional RIM objects.

You create additional RIM objects for the following reasons:

- To increase the number of connections to the RDBMS. The maximum number of connections per RIM object is 16.
- To distribute the work performed by RIM objects across multiple systems. For example, you can create RIM objects on separate managed nodes, so that the processing required by the RIM objects is divided between the two systems.

Creating RIM Objects

You create RIM objects using the **wcrtrim** command. To create RIM objects for the inventory data handler, you must specify the application label of the RIM object using the **–a** option. You must also configure the maximum number of connections that the RIM object is allowed to establish with the configuration repository using the **–m** option.

Note: The **wcrtrim –a** and **–m** options are currently used only for Tivoli Inventory RIM objects. Therefore, these options are not described in the Tivoli Management Framework documentation.

For example, to create a RIM object named invdh_2 and set the maximum number of RDBMS connections to 10, run the following command:

```
wcrtrim -v Informix -h bernard -d inventory \
-H /data/rdbms/informix -u informix -s az914shm \
-a invdh -m 10 invdh_2
```

where:

–v Informix Specifies the name of the database vendor that the RIM object represents.

–h bernard Specifies the name of the managed node on which the RIM object resides.

–d inventory Specifies the name (database ID) of the database to which the RIM object connects.

–H /data/rdbms/informix
Specifies the path to the database home directory.

–u informix Specifies the name of the database user that the RIM object uses.

–s az914shm	Specifies the server ID for the database.
–a invdh	Specifies that the application label for the RIM object is invdh. For RIM objects that the inventory data handler uses to connect to the RDBMS, you must set this value to invdh.
–m 10	Specifies the maximum number of connections that the RIM object has to the database. Set this option to a positive value from 1 to 16. The default value is 1. It is recommended that the total number of RDBMS connections set for all RIM objects used by the inventory data handler match the number of output threads set for the inventory data handler.
invdh_2	Specifies the name of the RIM object.

Configuring RIM Objects

This section describes how to determine the current settings for the RIM objects used by the inventory data handler, and how to change those settings.

You can view and set most options for RIM objects using the **wgetrim** and **wsetrim** commands. For more information about these commands, see the *Tivoli Management Framework Reference Manual*. However, to view and set the values for the application label and maximum number of database connections, you must use the following commands. First, determine the object ID (OID) of the RIM object by running the following command:

```
wlookup RIM_object_name
```

where *RIM_object_name* is the name of the RIM object, for example invdh_1.

The output from this command will be similar to the following example:

```
invdh_1 1228633823.2.20#RIM::RDBMS_Interface#
```

In this example, `1228633823.2.20#RIM::RDBMS_Interface#` is the OID for the RIM object invdh_1. To view the application label for this RIM object, run the following command:

```
idlcall OID _get_application_type
```

where *OID* is the OID for the RIM object, in this example 1228633823.2.20#RIM::RDBMS_Interface#.

For RIM objects that the inventory data handler uses, this value should be `invdh`. For other RIM objects, this value should be null ("").

To change this value, run the following command:

```
wsetrim -a value RIM_object_name
```

To view the maximum number of RDBMS connections set for this RIM object, run the following command:

```
idlcall OID _get_max_conn
```

where *OID* is the OID for the RIM object.

To change this value, run the following command:

```
wsetrim -m 10 RIM_object_name
```

Set this option to a positive value from 1 to 16. The default value is 1.

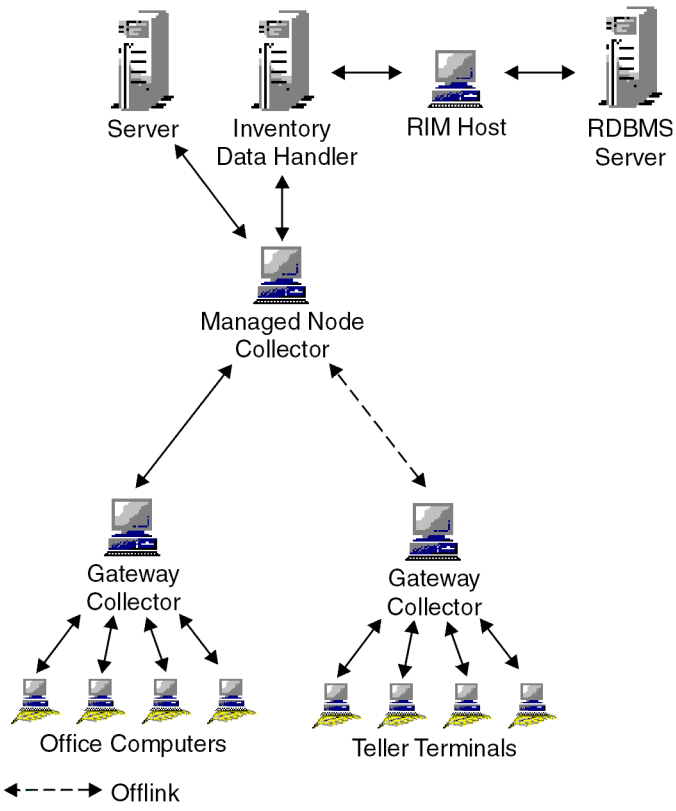
For more information about the RIM objects installed with Tivoli Inventory, see "Changing Passwords for RIM Objects" on page 2-28.

Scheduling Collections

To schedule collections, you specify that a collector should not collect data from certain downstream collectors at a particular time. Scheduling collections can also be referred to as scheduling *offlinks*. Offlinks are links to collectors from which you do not want to collect data.

Note: You can specify offlinks only between collectors. You cannot specify offlinks between collectors and endpoints.

You can control the time when collections are transferred through the collection hierarchy by specifying the times when the links between collectors are turned on and off. For example, a bank may have a network that contains teller terminals as well as several other computers. Because the teller terminals must have quick access to bank records on the network server during business hours, the bank may choose to prohibit collections from the teller terminals during these hours. See the following figure for an illustration of this example.



When you turn off a link between collectors, the CTOC for a pending collection moves to the deferred queue of the downstream collector. When the link between the collectors is restored, the CTOC in the deferred queue is processed.

You schedule collections by defining tasks and jobs in the Tivoli Management Framework and then scheduling the jobs with the Tivoli scheduler. To do this, you must be familiar with the concepts described in the chapter about task libraries in the *Tivoli Management Framework User's Guide*. The following is an overview of the procedure to schedule collections.

1. Create a task that turns off links to a collector, then halts and restarts the collector so the changes take effect:

   ```
   wcollect -x offlinks_range_to_prohibit_collection collector
   wcollect -h immediate
   wcollect -s
   ```

   where:

   offlinks_range_to_prohibit_collection
   : Specifies the object dispatcher numbers of the collectors for which links to the specified collector must be turned off.

   collector
   : Specifies the collector to which links must be turned off.

   To get the object dispatcher number of a collector, use the **odadmin** command and **odlist** option. For more information, see the *Tivoli Management Framework Reference Manual*.

   You can list the object dispatcher numbers in ascending numeric order, separated by commas, as shown in the following example:

   ```
   wcollect -x "4,5,6,7" collector
   ```

   Or, you can use a dash to indicate a range of object dispatcher numbers:

   ```
   wcollect -x "4-7" collector
   ```

   You must enclose the range of offlinks in double quotation marks ("").

   See “wcollect” on page B-8 for more information about the **wcollect** command and **–x** option.

2. Create a task that turns on the links to all systems for which links were previously turned off, then halts and restarts the collector so the changes take effect:

   ```
   wcollect -x "" collector
   wcollect -h immediate
   wcollect -s
   ```

3. Repeat steps 1 and 2 on each collector for which you want to schedule collections.

4. Create jobs to run these tasks.

5. Use the Tivoli scheduler to control when and how often to run these jobs.

You can also schedule collections by creating a job that starts and stops collectors using the **wcollect –s** and **–h** options. Alternatively, you can schedule a job that shuts down a collector's output queue or input queue by setting the **wcollect –o** or **–t** options for that collector to 0.

Note: When you reconfigure a collector, the changes do not take effect until you restart the collector.

Controlling the Flow of Network Data

SCS provides features to control the flow of SCS data across your network. These features are helpful if you have slow network links or if you want to specify when SCS data crosses your network. SCS provides the following mechanisms to control data flow:

- **Offlinks**—Offlinks are the primary way to regulate SCS traffic across your network. You can use offlinks to enable and disable SCS traffic between collectors at specified times. To use offlinks, you install a collector on either side of the network that requires flow control, then schedule offlinks using the Tivoli scheduler. For more information about scheduling offlinks, see "Scheduling Collections" on page 4-14 and "wcollect" on page B-8.
- **Transmission chunk size**—You use the **wcollect –c** option to configure the size of transmission chunks. This option enables you to control the size of the SCS data packet that crosses the interobject message (IOM) connection. The default transmission chunk size is 1 MB. If you specify a smaller size, the application data send is sent in smaller fragments. Decreasing transmission chunk size might be beneficial for slow links because the link is not congested with large block transmissions of SCS data. You configure transmission chunk size on the downstream collector. For more information about configuring transmission chunk size, see "Configuring a Collector" on page 4-5 and "wcollect" on page B-8.

  Note: Offlinks and transmission chunks affect data transmission between collectors, or between a collector and the inventory data handler. These mechanisms do not affect

transmissions between an endpoint and the gateway collector.

- **Input Threads**—Collectors use input threads to open an IOM session to a downstream collector for retrieving data. You can specify the maximum number of input threads for a collector using the **wcollect –t** command. You can increase the maximum number of input threads to allow more concurrent SCS IOM sessions, or decrease the number to reduce SCS IOM traffic coming into the collector. For more information about configuring a collector's input threads, see "Configuring Threads" on page 4-7 and "wcollect" on page B-8.

  Note: By configuring a gateway collector's input threads, you can help to reduce the load on the gateway by limiting the number of simultaneously active SCS sessions to endpoints from that gateway. This also helps reduce SCS network traffic coming into the gateway.

Obtaining Information about SCS

Using various SCS commands, you can view the following information:

- The configuration and status of a collector
- A description of the CTOCs on a collector
- The contents of a collector's queues
- The configuration of the inventory data handler
- Details about current inventory scans
- Details about SCS and Tivoli Inventory status collector options of an inventory profile

This section describes the commands you use to get information about SCS, and provides information about viewing inventory data stored in the configuration repository. For specific information about using these commands and their options, see Appendix B, "Commands."

Collector Configuration

Use the **wcollect** command to get information about the current configuration of a collector. You can view the level of debugging information being written to the collector log file, maximum size of the log file, location of the run-time directory, depot size, size of data transmission chunks, thread settings, collection retry settings, offlinks, and other information.

To view configuration information about a collector, enter the following command:

```
wcollect collector
```

where *collector* is of the form **@ManagedNode:***collector_name*, **@Gateway:***collector_name*, or **@InvDataHandler:inv_data_handler**.

Collector Status

Use the **wcstat** command to get status information about a collector. You can view this information for assistance in troubleshooting collectors.

To view status information about a collector, enter the following command:

```
wcstat collector
```

where *collector* is of the form **@ManagedNode:***collector_name* or **@Gateway:***collector_name*.

CTOCs and Queues

Use the **wcstat** command to get information about a CTOC or one or more of a collector's queues.

Information about a CTOC includes the CTOC ID, priority of the scanned data, collector status, the source and destination of the scan data, the method to be called on the destination object, client properties, collection status, and number of retries. To view information about a CTOC, enter the following command:

```
wcstat -v ctoc_id collector
```

where *ctoc_id* specifies the CTOC for which you want status information, and *collector* is of the form **@ManagedNode:***collector_name* or **@Gateway:***collector_name*.

A collector has input, output, error, completed, and deferred queues. You can view the following information about each of these types of queues: CTOC ID, priority of queued data, collector status, source and destination information, the method to be called on the destination object, scan ID, collection status, and the number of times that this collector has retried submitting requests to an upstream collector.

To view information about a collector's queues, enter the following command:

```
wcstat -q [ioecd] collector
```

where **[ioecd]** specifies the queue or queues for which you want information (input, output, error, completed, or deferred, or any combination), and *collector* is of the form **@ManagedNode:***collector_name* or **@Gateway:***collector_name*.

Inventory Data Handler Configuration

The **wgetinvdh** command returns configuration information about the inventory data handler, including scan completion notification and the number of retries and the timeout period for attempts to write data to a RIM object. The **–a** option causes the **wgetinvdh** command to return all available information about the inventory data handler. You can use several other options to return specific information. See "wgetinvdh" on page B-34 for more information.

Inventory Scans

The **wgetscanstat** command returns details about inventory scans. If you run the **wgetscanstat** command without any options, it returns a list of scan IDs of currently pending inventory scans along with the names of the inventory profiles that initiated those scans. You can also use options that provide information about the success, failure, or pending status of specific scans or all current scans. See "wgetscanstat" on page B-61 for more information.

Inventory Profiles

The **wgetinvglobal** command returns information about status notification settings, distribution settings, and data settings for an inventory profile. If you run this command with no options, it returns all available information about these settings for the inventory profile. You can use other options that provide specific information. See "wgetinvglobal" on page B-37 for more information.

Data Stored in the Configuration Repository

To view inventory data stored in the configuration repository, you can use any of the predefined queries provided with Tivoli Inventory or create your own queries. For more information, see Chapter 8, "Querying Inventory Information."

Configuring Tivoli Inventory Status Information

You can configure Tivoli Inventory to return status information about scans. Status refers to the success or failure of a scan on a particular inventory profile target.

You use the **wsetinvdh** and **wsetinvglobal** commands to configure options for notification and logging of Tivoli Inventory status information:

- The **wsetinvdh –s** option specifies whether the inventory data handler stores status information that can be restored in case of a system failure.
- The **wsetinvglobal –f** option specifies the name of the log file where status information is stored. The **wsetinvglobal –h** option specifies the host upon which the log file is stored. These options apply only if the **wsetinvglobal –l** option specifies to send notification to a log file.
- The **wsetinvglobal –n** option specifies when notification is sent. You can choose to send notification immediately, periodically, or when scans on all targets are complete.
- The **wsetinvglobal –t** option sets the circumstances under which a notice is sent for the specified profile. You can choose to send

notification when scans complete successfully, when scans fail, or for all targets.

- The **wsetinvglobal –l** option specifies the location to which notifications are sent. You can choose to send notification to the Tivoli Inventory notice group, to the log file, the TEC console, or any combination of those options. You can also choose not to send notification.

You use the **–n** and **–q** options of the **wcrtinvdh** or **wsetinvdh** commands, in combination with the **wsetinvglobal –n** option, to specify how to bundle status information.

You can use the **wgetscanstat** command to return status information about current inventory scans. You can use the **wgetinvglobal** command to return information about status collector options for a specified profile.

For detailed information about these commands and their options, see Appendix B, "Commands."

Considerations for Interconnected TMRs

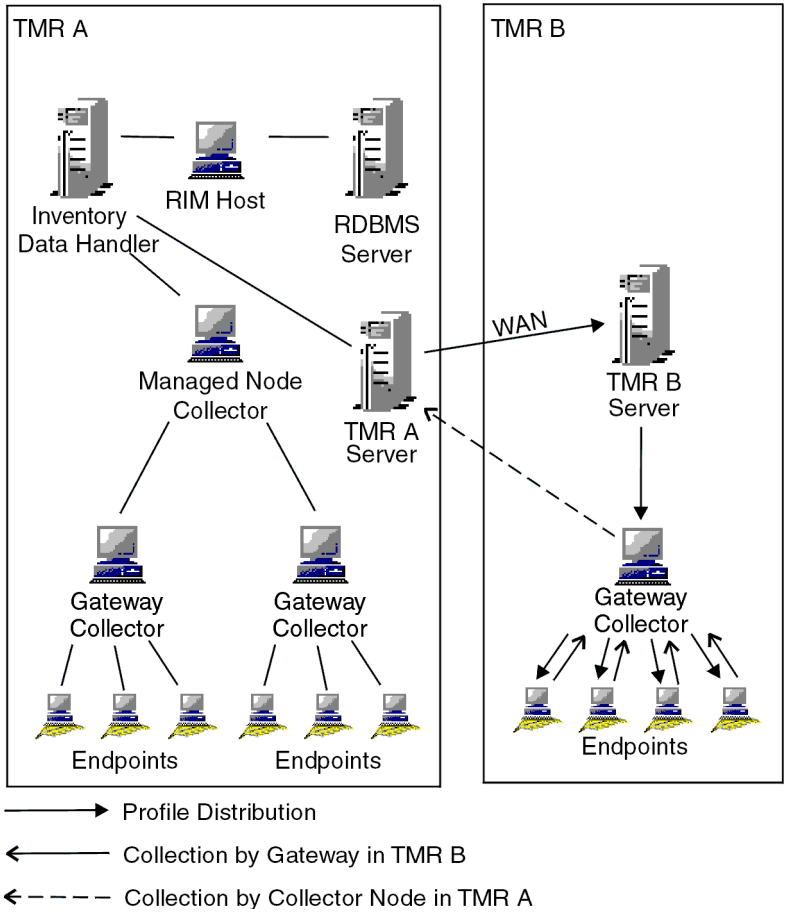
You can connect TMRs using either one- or two-way connections. However, to use SCS across the TMRs, you must configure a two-way connection between the TMRs.

When you connect TMRs, you must update resources. Connecting TMRs implies an initial exchange and periodic update of names and object identifiers contained in each server's name registry. See the *Tivoli Management Framework Planning for Deployment Guide* for more information about connecting TMRs and updating resources.

If you plan to run scans across TMRs and want Tivoli Inventory to return the data using SCS, you must install SCS in both TMRs.

SCS uses MDist 2 repeater hierarchies to obtain wide area network (WAN) entry point information. For example, you might have two TMRs connected by a WAN: TMR A and TMR B. TMR A contains the inventory data handler. A profile distributes from TMR A to endpoints in TMR B. The gateway for the endpoints in TMR B collects scan results from these endpoints. Next, the WAN-entry-point collector node in TMR A collects the scan results from the gateway in TMR B and sends

the data to the inventory data handler. The following figure illustrates this operation.



Where to Go from Here

Now that you have configured SCS, you are ready to collect and store information about the systems you are managing. You must create inventory profiles to do so. See Chapter 5, “Working with Inventory Profiles.”

5

Working with Inventory Profiles

This chapter describes the steps necessary to create, customize, clone, and delete inventory profiles. When setting up your inventory profiles, consider what information you need to gather, how you will use the information, and the constraints of your Tivoli environment. Once you have determined how you can best use the information gathered by Tivoli Inventory, you are ready to configure inventory profiles.

See Chapter 6, "Distributing Inventory Profiles," for more information about distributing inventory profiles and how Tivoli Inventory populates the configuration repository. See Chapter 7, "Collecting Custom Information with Tivoli Inventory," for information about customizing Tivoli Inventory to gather information that is not provided by default.

This chapter includes the following information:

- An explanation of how to use inventory profiles. (See "Using Inventory Profiles" on page 5-2.)
- Information about creating an inventory profile. (See "Creating an Inventory Profile" on page 5-3.)
- Information about setting the scanning instructions and other options in an inventory profile. (See "Customizing an Inventory Profile" on page 5-8.)
- Steps for creating an inventory profile by cloning an existing inventory profile. (See "Cloning an Inventory Profile" on page 5-25.)

- Steps for deleting an inventory profile. (See "Deleting an Inventory Profile" on page 5-27.)
- Steps for renaming an inventory profile. (See "Renaming an Inventory Profile" on page 5-29.)

Using Inventory Profiles

You can use an inventory profile to distribute a configuration file to an endpoint, scan for hardware and software data, or run a custom script or executable.

Tivoli Inventory provides several options for customizing inventory profiles. For example, you can create an inventory profile that collects different sets of data from PC and UNIX systems. This flexibility enables you to collect only the data you need from each platform. You can also specify the hardware data that you want to collect, and configure the files and directories to include or exclude during software scans.

Note: Tivoli Inventory does not scan NFS-mounted systems. On Windows NT systems, Tivoli Inventory does not scan mapped drives.

You can customize a profile to run a script on target systems. Scripts can be used to run optional scanning programs. For example, you could create and then run a script that gathers user information from a text file and then converts the data to Management Information Format (MIF). You can also update the BIOS dictionary file provided with Tivoli Inventory through a script that runs on your profiles. See Chapter 7, "Collecting Custom Information with Tivoli Inventory," for more information about scripts specific to BIOS.

You can control whether the inventory profile reads the MIF file generated by the scan. Using this option, you can create one inventory profile that scans the endpoint and then stores the scan data locally, and another inventory profile that sends the data to the configuration repository.

Saving the information in the RDBMS can be time-consuming if you choose to replace the old information with current scan results. You can, however, choose to save only the differences between the current scan and the previous scan. When you do this, Tivoli Inventory compares the

current MIF files to the MIF files generated from the previous scan and saves only the differences in the configuration repository.

Creating an Inventory Profile

When you create an inventory profile, you do so through a profile manager. Use profile managers to organize your profiles into logical groups. You can either create an inventory profile in an existing profile manager or create a new profile manager. For example, if you want to use profile managers to organize Tivoli profiles according to function, create a new profile manager named Inventory Profiles for all the inventory profiles in a policy region. On the other hand, if you have profile managers that organize endpoints according to operating system, you could create an inventory profile in each profile manager. Each profile could include a script or executable and scan instructions that are specific to the operating system of the systems in the profile manager.

When you create an inventory profile for distribution to endpoint subscribers, make sure that the profile manager is in Dataless Endpoint Mode. When you create an inventory profile for distribution to profile manager subscribers, make sure that the Dataless Endpoint Mode option is deselected. Make sure that the subscribers of the profile manager are appropriate targets for the profile. The only valid targets for an inventory profile are endpoints and profile managers. If the profile manager has managed nodes as subscribers, Tivoli Inventory ignores them.

For information about subscribing systems to profile managers, see the *Tivoli Management Framework User's Guide*.

When you name an inventory profile, there are several things to consider. First, if you are planning to have a large number of inventory profiles, choose a name for each profile that reflects its function. For example, a profile that is customized to scan only hardware information could be named Hardware_Scan.

The following table provides the authorization roles required to create an inventory profile:

Operation	Context	Role
Create an inventory profile	Profile manager	**senior** or **super**

You can create a profile from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to create an inventory profile from the Tivoli desktop:

1. In a policy region, add the **InventoryConfig** and **ProfileManager** resources to the list of managed resources.

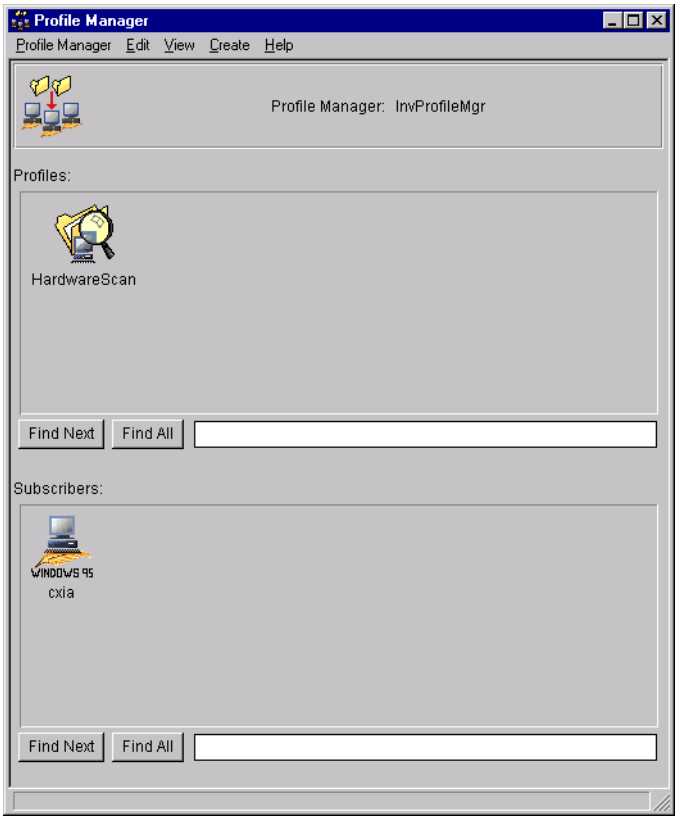
   For instructions about adding resource types and creating profile managers in a policy region, see the *Tivoli Management Framework User's Guide*.

2. Create or select a profile manager in which the inventory profile will reside.

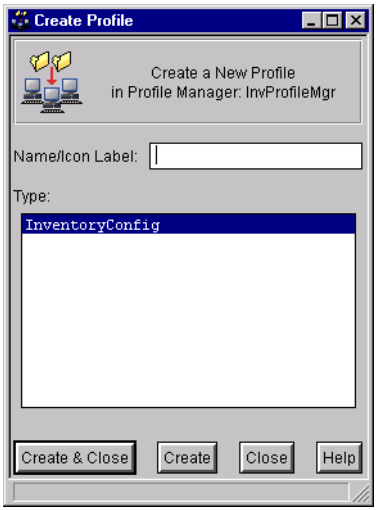
3. Subscribe endpoints and profile managers to which you want to distribute profiles from this profile manager.

   For more information about profile managers and subscribers, see the *Tivoli Management Framework User's Guide*.

4. From the policy region, double-click a Profile Manager icon. The **Profile Manager** window is displayed.


Profile Manager
Profile Manager Edit View Create Help
Profile Manager: InvProfileMgr
Profiles:
HardwareScan
Find Next
Find All
Subscribers:
WINDOWS 95
cxia
Find Next
Find All

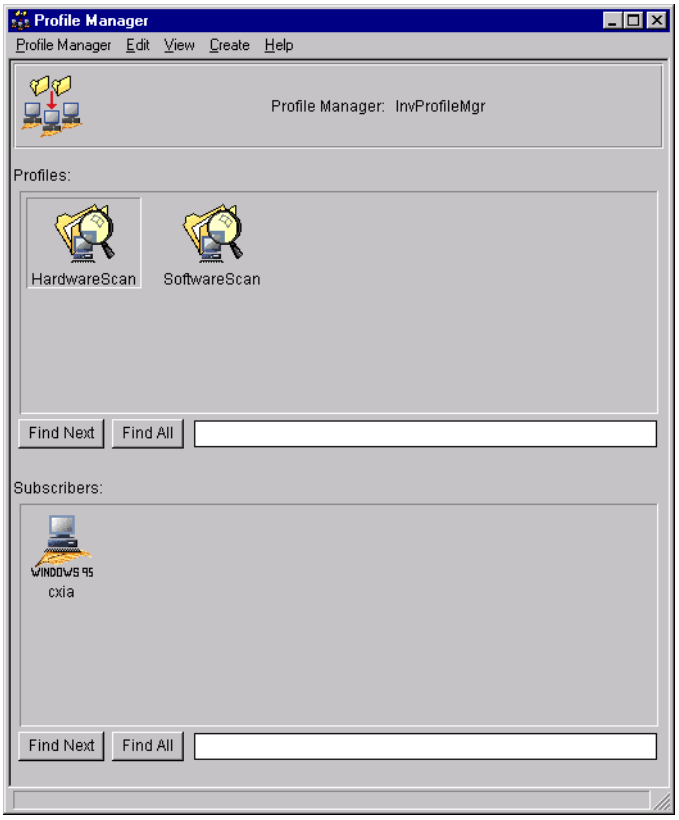
5. In the **Profile Manager** window, select **Profile** from the **Create** menu. The **Create Profile** dialog is displayed.



6. In the **Name/Icon Label** text box, enter a unique name for the profile.
7. From the **Type** scrolling list, select **InventoryConfig**.

   Note: If the **InventoryConfig** resource type is not available, you must add this resource type as a managed resource for your policy region. For more information on this procedure, see the *Tivoli Management Framework User's Guide*.
8. Click the **Create & Close** button to create the new profile and return to the **Profile Manager** window. An icon representing the new inventory profile is displayed in the **Profile Manager** window. The profile has the default scanning options. For

instructions on customizing a profile, see “Customizing an Inventory Profile” on page 5-8.



Command Line

Because inventory profiles must reside in a profile manager, you must create a policy region and profile manager in which to create the profile. To create policy regions and profile managers, see the *Tivoli Management Framework Reference Manual* for information on the **wcrtprf** and **wcrtprfmgr** commands.

To use the **wcrtprf** command to create the **Enterprise** profile in the **Jamaica** profile manager, enter the following:

```
wcrtprf @ProfileManager:Jamaica InventoryConfig \
Enterprise
```

where:

@ProfileManager:Jamaica

Specifies **Jamaica** as the name of the profile manager in which to create the profile.

InventoryConfig

Specifies the type of profile to create.

Enterprise

Specifies **Enterprise** as the name of the new profile.

For more information about using the command line to create a profile, see the **wcrtprf** command in the *Tivoli Management Framework Reference Manual*.

Customizing an Inventory Profile

You can customize profiles or sets of profiles to serve various purposes in your Tivoli environment. When you customize a profile, you specify what happens on target systems when the profile is distributed. For example, you can create an inventory profile to collect the complete hardware and software configuration on each system in your Tivoli environment. Then, you can create another profile for subsequent scans that updates the configuration repository with only new or changed data about the hardware and software on those systems.

When you customize inventory profiles, you can specify what happens on the endpoint when you distribute the profile, including the following:

- Whether to run a scan or just distribute a configuration file
- How to save scan data to the configuration repository (differences or current results)
- The type of scan to run (hardware, software, or both)
- Whether to read scan results and which scan results to read (hardware, software, or custom MIF files)
- Which files to scan
- Which directories to scan

- What information to collect: matching software signature information, header information (on PC systems only), or basic file information
- Whether to apply a custom filter to scans for basic file information
- Whether to generate checksum values for scanned files
- Whether to run scripts that you specify on the target systems

Customizing Global Properties

You set global properties for an inventory profile using the Global Properties window in the Tivoli Inventory graphical user interface (GUI). Global properties include the distribution and data options for a profile. These options apply globally to all scans that a profile distributes.

- **Distribution Options** specify what actions occur when you distribute a profile. You can either distribute the profile without running the scan, or distribute the profile and run the scan.
  - **Distribute configuration file** distributes the configuration file to the endpoint, but does not start the scan. Choose this option when you want to distribute the configuration file to an endpoint that will be scanned later. For example, you must distribute a configuration file to an endpoint before you initiate a scan from the endpoint using the **wepscan** command.
  - **Distribute configuration file and run scan** distributes the configuration file to the endpoint, and then runs the scan on the endpoint. Choose this option when you want to scan the endpoint immediately.
- **Data Options** specify how the data gathered by this profile is stored in the configuration repository:
  - **Update with differences** compares MIF files read during the current distribution with those from the previous distribution and saves the differences. Only data that has been added or changed since the last scan is stored in the configuration repository. Data from previous scans that is not present in the current scan is deleted from the configuration repository. No record of changes is kept unless history tables are used. Select

this option following an initial scan to reduce the amount of information that is sent to the configuration repository.

- **Replace with current results** replaces existing data in the configuration repository with the data gathered by this scan. Select this option when performing an initial scan and when you want to save a fresh image of the hardware and software on the target systems.

If you are using history tables, it is recommended that you use the Replace with current results option for the first scan, and then use Update with differences for subsequent scans. For more information about history tables, see "History Tracking" on page C-2.

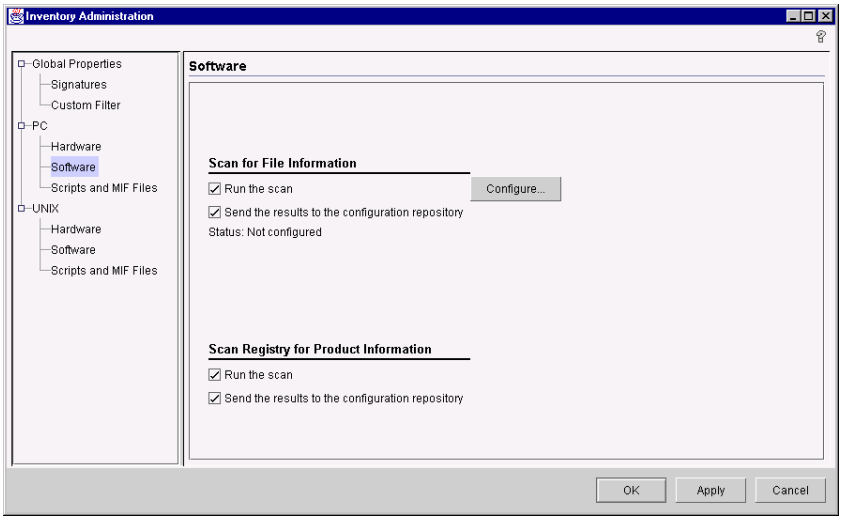
For more information about the Global Properties window, see the Tivoli Inventory online help.

Customizing a Software Scan

You customize software scans from the desktop using the Software window.

Software Scan Options for PC

To customize software scans of PCs, select **PC–>Software** in the left pane of the Inventory Administration window. The Software window is displayed.



From this window, you can enable and configure scans for file information and scans of the registry for product information.

- **Scan for File Information** scans the files stored on the local hard drive or drives of an endpoint. Using a scan for file information, you can collect such data as file name; file size; path; permissions; and creation, modification, and access dates. You can generate checksum values for scanned files. You can also filter the scans by including or excluding files, file types, or directories. To configure this scan, click **Configure**, and then set the options on the Software Scan Configuration screen. For more information, see "Software Scan Configuration Options" on page 5-14.

  Status indicates whether you have changed the options on the Software Scan Configuration screen during the current editing session of the profile.

  To enable the **Scan for File Information**, select one or both of the following options:

  - **Run the scan** scans the endpoint, and then stores the data on the endpoint in a MIF file.

- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

  You can select both **Run the scan** and **Send the results to the configuration repository** to create a profile that scans an endpoint and then immediately sends the data to the configuration repository. Or you can create two profiles, one with only **Run the scan** selected and the other with only **Send the results to the configuration repository** selected. Doing this enables you to scan the endpoint and store the data to be collected at a later time. For example, you might choose to scan endpoints during business hours, and then send the data to the configuration repository during off-peak hours when more network bandwidth is available.

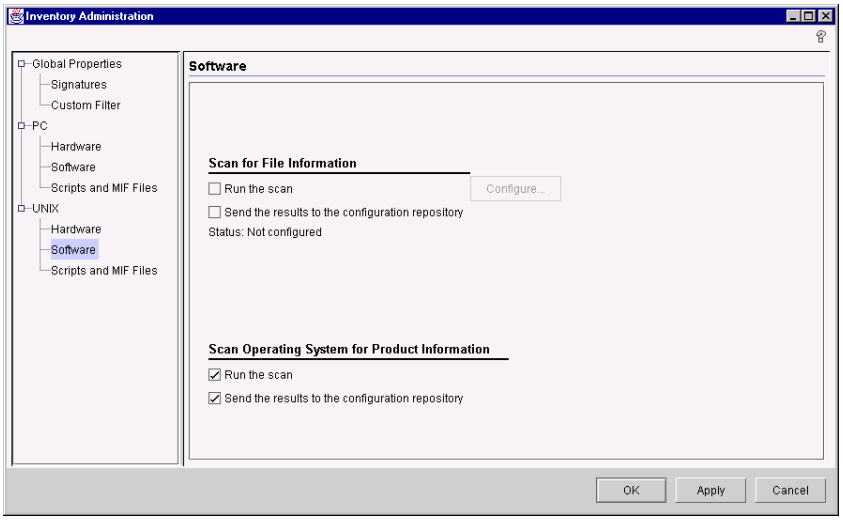
■ **Scan Registry for Product Information** scans the registry of endpoints on supported Windows platforms for information about installed products. To enable the **Scan Registry for Product Information**, select one or both of the following options:

- **Run the scan** scans the endpoint, and then stores the data on the endpoint in a Management Information Format (MIF) file.
- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

Note: This option has no effect on NetWare and OS/2 endpoints.

Software Scan Options for UNIX

To customize software scans of UNIX systems, select **UNIX–>Software**. The Software window is displayed.



From this window, you can enable and configure scans for file information and scans of the operating system for product information.

- **Scan for File Information** scans the files stored on the local hard drive or drives of an endpoint. Using a scan for file information, you can collect such data as file name; file size; path; group; owner; permissions; creation, modification, and access dates; and, optionally, checksum values. You can also filter the scans by including or excluding files, file types, or directories. **Status** indicates whether you have configured the scan for file information. To configure this scan, click **Configure**, and then set the options on the Software Scan Configuration screen. For more information, see “Software Scan Configuration Options” on page 5-14.

  To enable the **Scan for File Information**, select one or both of the following options:

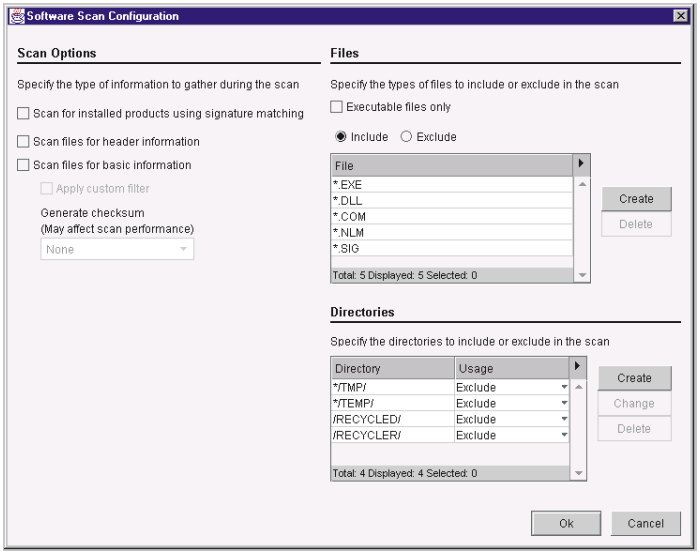
  - **Run the scan** scans the endpoint, and then stores the data on the endpoint in a Management Information Format (MIF) file.
  - **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

- **Scan Operating System for Product Information** scans the UNIX operating system for information about installed products and patches. To enable this scan, select one or both of the following options:
  - **Run the scan** scans the endpoint, and then stores the data on the endpoint in a Management Information Format (MIF) file.
  - **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

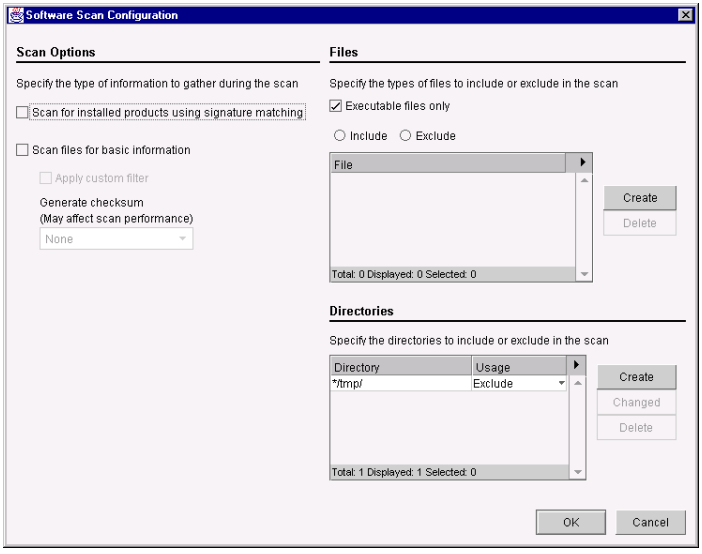
Software Scan Configuration Options

The Software Scan Configuration dialog box enables you to further customize your software scan options. You access the Software Scan Configuration dialog box by clicking Configure from the Software window. Using this dialog box, you can specify the type of information that the scan gathers and specify the files and directories to include or exclude during the scan.

The following figure shows the Software Scan Configuration dialog box for PCs:



The following figure shows the Software Scan Configuration dialog box for UNIX systems:



The following sections describe the options in the Software Scan Configuration dialog box.

Scan Options

Under **Scan Options**, specify the type of information that you want to collect. Each option uses a different method to collect data.

- **Scan for installed products using signature matching**. In this type of scan, the list of software signatures is distributed to the endpoint. This list is compared to the files scanned on the endpoint. When a file matches a software signature, the software signature data for that file is sent to the configuration repository. Software signature data includes the name, size, and usually the Quick checksum value of the file used to identify the software product. This identifying file is usually the primary executable file for the product, for example notepad.exe. Custom software signatures may return additional data.

  Software signature scans do not return data for all files on an endpoint. They return data only for files that match a software

signature. Because only data for matching files is sent, this scan returns less data than scans for basic file information. Therefore, it reduces the amount of data that you must send through the network and store in the configuration repository. Also, because each endpoint performs the processing necessary to compare the software signatures to the scanned files, this scan is more efficient than performing signature matching at the configuration repository using a scan for basic file information. For more information about software signatures, see Chapter 7, Collecting Custom Information with Tivoli Inventory.

- **Scan files for header information**. A header is a space within an executable or dynamic link library file used to store data about the application to which the file belongs. This option searches the header of scanned files for header information. Microsoft Windows headers usually contain the following data: company name, product name, and product version. This option returns information for all scanned files that contain header information, not just files matched with software signatures. This option is not available for UNIX software scans. This option has no effect on scans of NetWare and OS/2 endpoints.
- **Scan files for basic information**. This option collects the name and path of scanned files. Other information, such as date of creation, modification, and last access, is collected on some platforms. Unlike software signature scans or scans for header information, this type of scan can collect information about all files on each scanned endpoint. Or, you can filter the scan to collect information about only the files you specify.

  The Apply custom filter check box enables you to filter scans for basic information. When you select this check box, scans for basic information are limited to the files listed in the Custom Filter dialog box. See the Tivoli Inventory online help for more information about the Custom Filter dialog box.

  The Generate checksum option allows you to collect the checksum value of scanned files. Checksum algorithms map a file's contents to a single fixed value. By comparing these values, you can determine whether a file's contents have changed over time, possibly due to data corruption or a virus. For example, you could

gather the MD5 checksum value for all files in your enterprise, and then monitor this value for changes to detect "Trojan horse" programs. The checksum options are as follows:

- The **None** option disables the checksum feature for the inventory profile you are customizing.
- The **Quick CRC** option uses the CRC-32 algorithm to produce a 32-bit value based on the first 1 KB of each file. This is the quickest checksum option, but produces the least reliable value.
- The **Full CRC** option uses the CRC-32 algorithm to produce a 32-bit value based on the full contents of each file. This option takes longer than the Quick option, but produces a more reliable value.
- The **MD5** option produces a 128-bit value based on the full contents of each file. The MD5 option provides a more reliable value than the Quick or Full options, but takes longer to complete.

Files

Use the Files scrolling list to specify the file names or file types that you want to include or exclude during the scan. To reduce scan time, limit the files that the profile will scan. You can provide specific file names to include or exclude, or you can include or exclude file types by providing a file extension such as *.exe. This list applies to scans for matching software signatures, scans for header information, and scans for basic file information.

Select the Executable files only option to scan only executable files. For Windows systems, an executable file is defined as a file with one of the following extensions: .EXE, .COM, .CMD, or .BAT. For UNIX systems, an executable file is defined as a file that has the execute permission set.

Directories

In the Directories scrolling list, specify the directories that you want to include in the scan, those that you want to exclude from the scan, or both. To reduce scan time, limit the directories that the profile will

search. You can list specific directories to exclude, or enter a partial path to exclude all subdirectories below a specified directory. This list applies to scans for matching software signatures, scans for header information, and scans for basic file information.

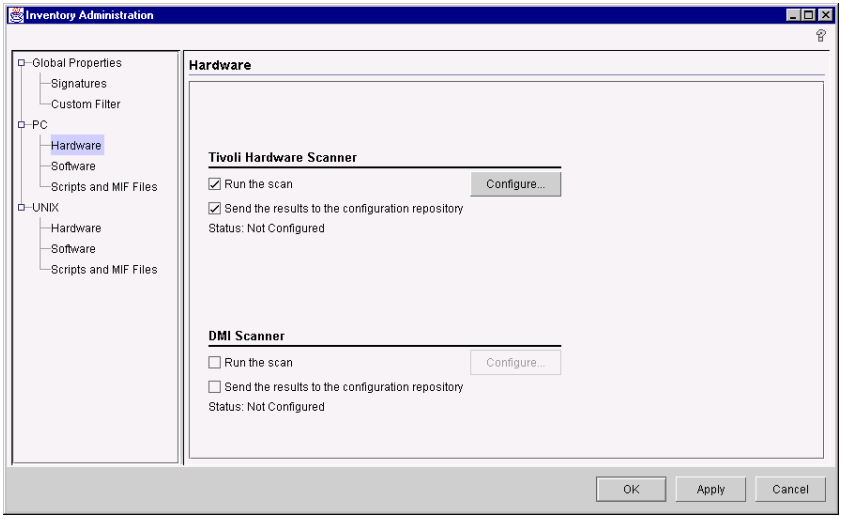
For more information about the Software Scan Configuration dialog box, see the Tivoli Inventory online help.

Customizing a Hardware Scan

By default, Tivoli Inventory scans PC and UNIX systems for a number of hardware components. However, you can customize an inventory profile to scan only the hardware components that you select. Moreover, for PC hardware scans, you can configure the inventory profile to collect hardware information from the Desktop Management Interface (DMI) layer of a PC, and you can select the DMI attributes to include in the scan. You customize hardware scans from the desktop using the Hardware window.

PC Hardware Scans

To customize hardware scans of PCs, select **PC–>Hardware** in the left pane of the Inventory Administration window. The Hardware window is displayed.



From this window, you can enable and configure the Tivoli hardware scanner, the DMI scanner, or both scanners.

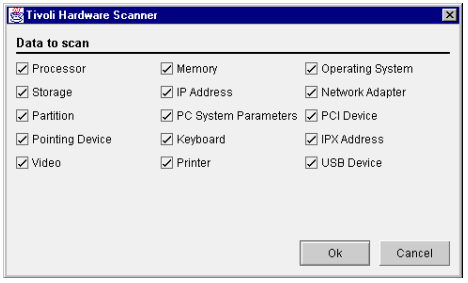
Tivoli Hardware Scanner

The Tivoli hardware scanner uses a combination of hardware, system, and registry calls to obtain characteristics about hardware components. On some Windows operating systems, the scanner incorporates Common Information Model (CIM) and Windows Management Instrumentation (WMI) support to obtain standards-based information.

To enable this scan, select one or both of the following options:

- **Run the scan** scans the endpoint, and then stores the data on the endpoint in a MIF file.
- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

Tivoli Inventory allows you to specify which hardware attributes that you want to scan for. For example, you could create an inventory profile that scans only for installed memory. To customize the list of hardware data collected by the Tivoli hardware scanner, click **Configure**. The Tivoli Hardware Scanner dialog box is displayed.



For more information about customizing PC hardware scans, see the Tivoli Inventory online help.

DMI Scanner

The DMI scanner scans the DMI layer of a PC for hardware information. DMI is an industry standard for describing and accessing information about PCs and PC components. Some PCs contain a DMI service layer,

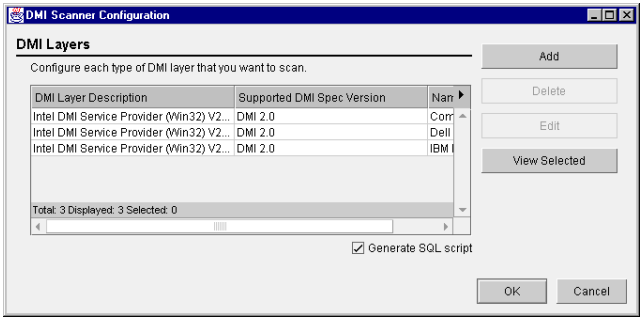
which contains information about the PC on which it is installed. Using DMI standards, you can manage your PC hardware independent of a specific computer, operating system, or management tool.

For example, you can identify the serial number on systems from any manufacturer that has implemented the DMI service layer on their computers. Depending on the manufacturer, the DMI service layer also might contain information about the cost of ownership associated with your hardware.

When you run the DMI scanner, Tivoli Inventory retrieves the attributes from the DMI service layer installed on the target systems and creates a MIF file, dmiscan.mif, that contains that information. Tivoli Inventory then reads the MIF file and populates the configuration repository accordingly.

To configure a scan for DMI information, you must complete the following steps:

1. In the Hardware window, enable the DMI scan by selecting **Run the scan**, and if desired, **Send the results to the configuration repository**:
   - **Run the scan** scans the endpoint, and then stores the data on the endpoint in a Management Information Format (MIF) file.
   - **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.
2. Click **Configure** to access the DMI Scanner Configuration dialog box.



Use this dialog box to configure the DMI options for a PC hardware scan. To scan a DMI layer, you must create a list of the hardware attributes that you want to scan for. To create this list, you retrieve and then edit each type of DMI layer that you want to scan. For more information about using this dialog box, see the Tivoli Inventory online help.

Notes

- Before you attempt to retrieve a DMI layer from an endpoint, make sure that you have distributed an inventory profile at least once to that endpoint. An inventory profile, no matter how it is configured, distributes the following files to the endpoint: dmiscan.exe, wcdmi.dll and wdmiutil.dll. These files must be present on the endpoint before you can retrieve the DMI layer.
- You must retrieve and configure each type of DMI layer that you want to scan. DMI layers vary by manufacturer and model.

3. Select the **Generate SQL script** check box in the DMI Scanner Configuration dialog box. When you select this option, Tivoli Inventory automatically generates an SQL script that you can use to create custom tables in the configuration repository for the DMI scan information.
4. Click **OK** in the DMI Scanner Configuration dialog box to generate the script. This script is named dmi_*RDBMS_type*_schema.sql, where *RDBMS_type* is the type of RDBMS that holds the configuration repository. The script is stored in the $BINDIR/TME/INVENTORY directory on the Tivoli Management Region (TMR) server. You must run this script before you distribute a profile that sends DMI scan data to the configuration repository.
5. To run the script in the configuration repository, make a client connection (for example, an SQL*Plus session) to the database server where the Tivoli Inventory configuration repository resides, and then execute the script. For more information about running a script from a client connection, see Chapter 3, “Creating the Configuration Repository.” This chapter provides instructions for

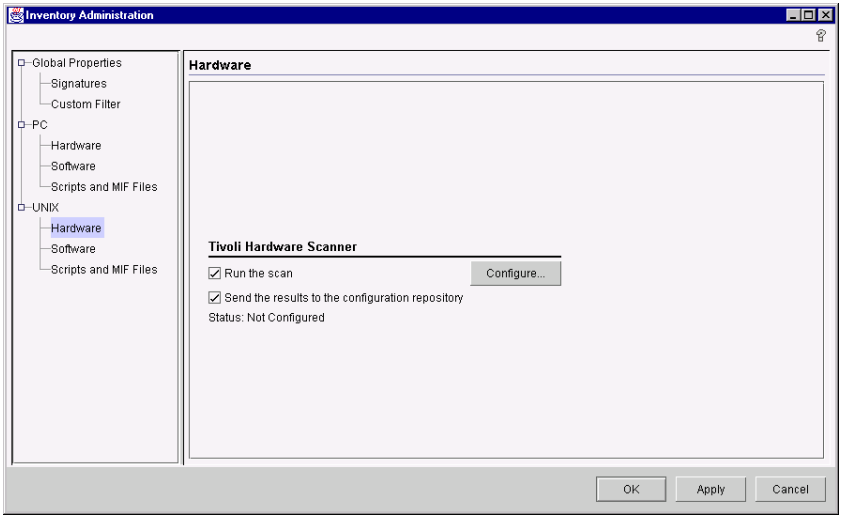
installing the configuration repository schema. Separate sections are provided for each type of supported RDBMS. Use the section appropriate to your configuration repository. Follow the instructions in this section, but use the dmi_*RDBMS_type*_schema.sql script instead of the inv_*xxx*_schema.sql script. When you run this script, watch for error messages that indicate the script did not complete properly.

6. Distribute the profile to the target systems.

For more information about configuring DMI scan options, see the Tivoli Inventory online help. For help with troubleshooting DMI scans, see Appendix E, "Troubleshooting."

UNIX Hardware Scans

To customize hardware scans of UNIX systems, select **UNIX–>Hardware**. The Hardware window is displayed.

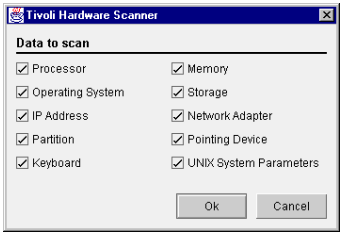


From this window, you can enable and configure the Tivoli hardware scanner. To enable this scan, select one or both of the following options:

- **Run the scan** scans the endpoint, and then stores the data on the endpoint in a MIF file.

- **Send the results to the configuration repository** collects the MIF file from the endpoint and sends the data to the configuration repository.

To customize the list of hardware data collected by the Tivoli hardware scanner, click **Configure**. The Tivoli Hardware Scanner dialog box is displayed.



For more information about customizing UNIX hardware scans, see the Tivoli Inventory online help.

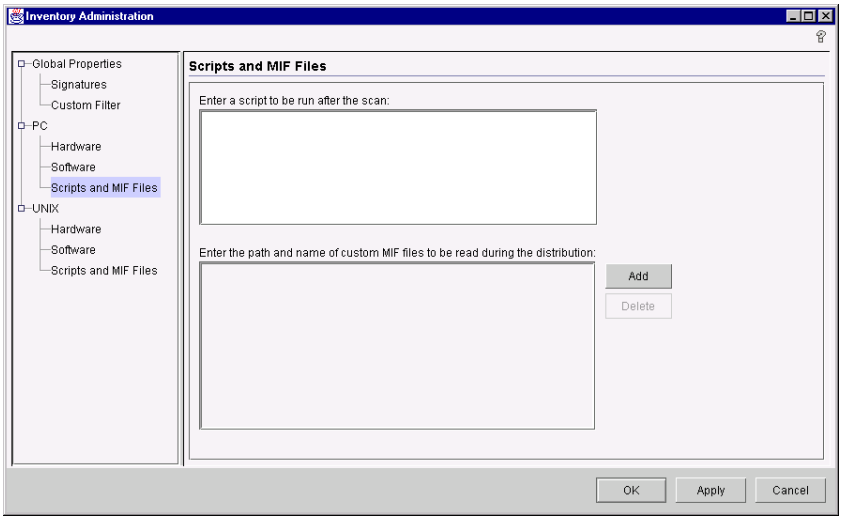
Using Custom Scripts and MIF Files

You can customize an inventory profile to run a custom script during a profile distribution. Scripts are run after the scan completes but before the MIF files are read. You can use this feature to generate a custom MIF file or to modify a MIF file created during a scan. See Chapter 7, "Collecting Custom Information with Tivoli Inventory," for instructions about using custom MIF files.

You must use scripts that are specific to the operating system of the target systems. For example, you might write a shell script for UNIX targets, a .BAT file for Windows targets, or a .CMD file for OS/2 targets. Include the directories of any commands or files that you reference in the script.

Note: You cannot run a script on NetWare endpoints.

To add a custom script or MIF file for PC systems, select **PC->Scripts and MIF Files**. To add a custom script or MIF file for UNIX systems, select **UNIX->Scripts and MIF Files**. The Scripts and MIF Files window is displayed.



In the upper text box, type or paste in a batch or command script that you want to run after an inventory scan. Enter the entire script in this field, not the path to a script file.

In the lower text box, list any custom MIF files that you want to be read during a distribution. To add a file to the list, click **Add**. To delete a file from the list, click the file in this text box, and then click **Delete**.

For more information about using scripts and custom MIF files, see the Tivoli Inventory online help.

Authorization Roles

You can customize an inventory profile from either the Tivoli desktop or the command line. The following table provides the authorization roles required to customize an inventory profile:

Operation	Context	Role
Customize a profile	Tivoli inventory profile	**Inventory_edit**, **admin**, **senior**, or **super**

Note: To customize an inventory profile from the Tivoli desktop, a user with the **Inventory_edit** role must also have the **user** role in the policy region in which the profile resides.

Desktop

See the Tivoli Inventory online help for the procedure to customize an inventory profile from the Tivoli desktop.

Command Line

To customize an inventory profile using the command line, you can first get information about the profile's current settings using one or more of the following commands: **wgetinvglobal**, **wgetinvpcfiles**, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixhw**, and **wgetinvunixsw**. You can then use one or more of the following commands to change the settings: **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvpcsw**, **wsetinvunixfiles**, **wsetinvunixhw**, and **wsetinvunixsw**. See Appendix B, "Commands," for more information about these commands and how to use them to customize inventory profiles.

Cloning an Inventory Profile

You can make a copy of an inventory profile by cloning it. (You must rename the profile.) This feature is especially useful if you have PCs in your Tivoli environment that have files residing on different drives. You can clone the original profile and change the drive letter for the files to be scanned.

The following table shows the context and authorization roles required to perform this task:

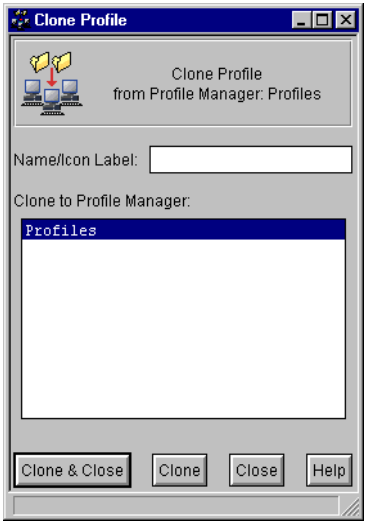
Operation	Context	Required Role
Clone a profile	Profile manager	**senior** or **super**

You can perform this task from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to clone an inventory profile from the Tivoli desktop:

1. From a profile manager, select the profile you want to clone.
2. Select **Profiles->Clone** from the **Edit** menu. The **Clone Profile** dialog box is displayed.



3. Enter the name for the new profile in the **Name/Icon Label** field.
4. Select the profile manager in which the new profile will reside from the **Clone Profile Manager** list.
5. Click the **Clone & Close** button to create the clone of the profile and return to the **Profile Manager** window.

Command Line

To use the **wcrtprf** command to clone the **UNIX_First_Scan** inventory profile, enter the following command:

```
wcrtprf -c @UNIX_First_Scan \
@ProfileManager:First_Scan_Profiles InventoryConfig UNIX_Update
```

where:

-c @UNIX_First_Scan
: Specifies **UNIX_First_Scan** as the profile to clone. You must use the at sign (@) when specifying the profile name.

@ProfileManager:First_Scan_Profiles
: Specifies **First_Scan_Profiles** as the profile manager to contain the new inventory profile.

InventoryConfig
: Specifies the type of profile to clone.

UNIX_Update
: Specifies **UNIX_Update** as the name of the new profile. Do not use an object path (**/Regions/…**) or registered name (**@name**). Specify the new name.

For more information about using the command line to clone inventory profiles, see the **wcrtprf** command in the *Tivoli Management Framework Reference Manual*.

Deleting an Inventory Profile

When you delete a profile, it is removed from the Tivoli object database and its icon is removed from the profile manager.

Note: If you delete an inventory profile for which you have scheduled a distribution, you must delete the job from the scheduler. Deleting a profile does not automatically delete the job. See the *Tivoli Management Framework User's Guide* for more information about using the scheduler.

The following table shows the context and authorization roles required to delete a profile:

Operation	Context	Required Role
Delete an inventory profile	Profile manager	**senior** or **super**

You can delete an inventory profile from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to delete an inventory profile from the Tivoli desktop:

1. From a profile manager, select the icons of the profiles you want to delete and select **Profiles->Delete** from the **Edit** menu. The **Delete Profiles** dialog is displayed.

   Note: You cannot recover a deleted profile.

   Disregard the message “The original profiles AND all of their copies will be deleted” because it does not apply to inventory profiles.

2. To confirm the deletion, click the **Delete** button. Tivoli Inventory deletes the profile.

Command Line

To use the **wdel** command to delete the **UNIX_First_Scan** profile and remove its icon from the **First_Scan_Profiles** profile manager in the **Inventory** policy region, enter the following command:

```
wdel \
/Regions/Inventory/First_Scan_Profiles/UNIX_First_Scan
```

or

```
wdel @InventoryConfig:UNIX_First_Scan
```

where:

/Regions/Inventory/First_Scan_Profiles/UNIX_First_Scan

Specifies the object path of the inventory profile to delete. You can specify object paths or registered names of Tivoli objects such as profiles and profile managers.

For more information about the **wdel** command, see the *Tivoli Management Framework Reference Manual*. See Appendix B, "Commands," for more information about object paths.

Renaming an Inventory Profile

To rename an inventory profile, follow these steps:

1. Clone the profile. Name the cloned profile the desired name for the original profile.
2. Delete the original profile.

Where to Go from Here

After you create and customize inventory profiles, you are ready to scan the systems in your Tivoli environment. Refer to Chapter 6, "Distributing Inventory Profiles," for instructions on distributing an inventory profile to subscribers and an explanation of the information that Tivoli Inventory collects. See Chapter 7, "Collecting Custom Information with Tivoli Inventory," to learn more about customizing Tivoli Inventory.

6

Distributing Inventory Profiles

After you create and customize your inventory profiles, you can distribute them to endpoints in your Tivoli environment. This chapter describes the steps that you take to distribute inventory profiles from the Tivoli desktop and from the command line interface. If you have not yet created inventory profiles, see Chapter 5, "Working with Inventory Profiles," for instructions.

This chapter includes the following information about inventory scans:

- An explanation of what happens during an inventory scan and steps for distributing an inventory profile. (See "Distributing Inventory Profiles" on page 6-1.)
- Information about how to initiate a scan from an endpoint rather than distributing the scan from an inventory profile (See "Performing an Endpoint-Initiated Scan" on page 6-8)
- An explanation of how information is saved in the configuration repository. (See "Populating the Configuration Repository" on page 6-9.)

Distributing Inventory Profiles

You can use the Tivoli desktop or command line interface to distribute the inventory profile to the target machines.

Scanning the machines in your environment for the first time can be time-consuming. However, after you have scanned the machines once,

you can use an inventory profile that is set to compare the results of the current scan with the results of the previous scan.

If you choose Update with Differences on the Global Properties window, Tivoli Inventory saves only new or changed data to the configuration repository. Because differences constitute a smaller amount of data and can be saved in the configuration repository more quickly, subsequent scans can take significantly less time than an initial scan. Refer to Chapter 5, "Working with Inventory Profiles," for information about setting up scans.

Tivoli Inventory provides information about events and errors that result from an inventory profile distribution. By default, this information is sent to the Tivoli Inventory notice group. Using the **wsetinvglobal** command and **–l** option, you can configure Tivoli Inventory to send this information to a log file, the TEC console, the Tivoli Inventory notice group, or any combination of those locations. See Appendix B, "Commands," for more information about the **wsetinvglobal** command. See the *Tivoli Management Framework User's Guide* for more information about viewing group notices. The following list includes some of the situations in which Tivoli Inventory posts a notice:

- A scan is unsuccessful because an endpoint is not available.
- An RDBMS operation fails.
- A custom MIF file is not found in a specified location.
- A custom MIF file is not formatted correctly.

The following table provides the authorization roles required to distribute an inventory profile:

Operation	Context	Role
Distribute an inventory profile	Inventory profile	**Inventory_scan**, **senior**, or **super**

Note: To distribute an inventory profile from the Tivoli desktop, a user with the **Inventory_scan** role must also have the **user** role in the policy region in which the profile resides.

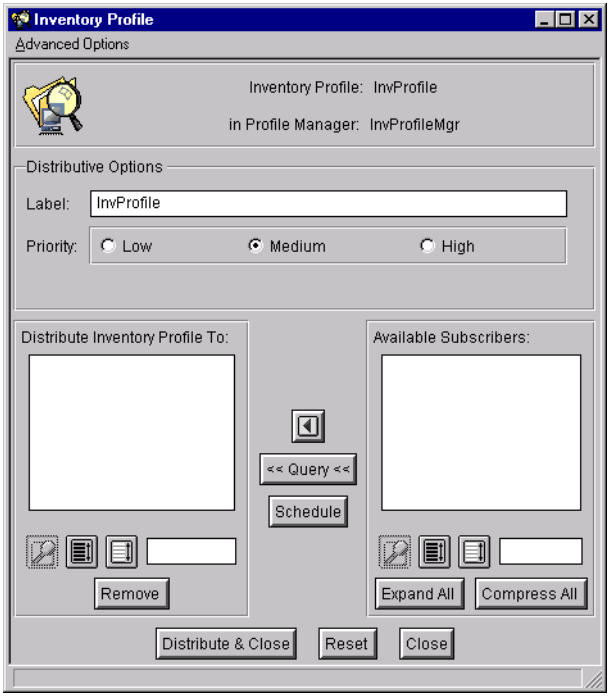
You can distribute a profile from the Tivoli desktop or the command line.

MDist 2 enables you to assign a priority to profile distributions. You can choose high, medium, or low priority. You control priority processing of distributions by assigning a number of connections that a repeater can open for distributions with a given priority. Moreover, if no connections are available for a given priority, the repeater tries to borrow a connection from a lower priority. For more information about assigning priorities to distributions and setting maximum priority connections, see the *Tivoli Management Framework User's Guide*.

Desktop

Complete the following steps to distribute an inventory profile from the Tivoli desktop.

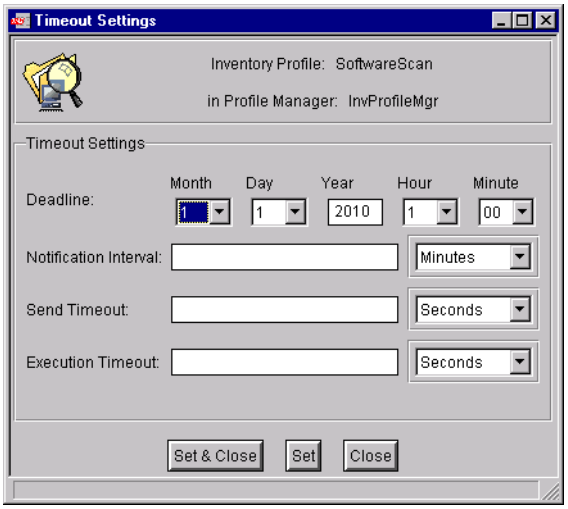
1. In a profile manager, right-click an icon for an inventory profile, and then select **Distribute**. The Inventory Profile dialog box is displayed.



2. Set the priority for the distribution by selecting the **Low**, **Medium**, or **High** radio button.
3. In the Available Subscribers scrolling list, select the machines or profile managers to which you want to distribute the profile. The Available Subscribers list shows the subscribers for the profile manager in which the profile resides.
4. Complete one of the following options:
   - Click the arrow to move targets between the Available Subscribers and Distribute Inventory Profile To scrolling lists.

- Press the **Query** button to select targets through a query. See Chapter 8, "Querying Inventory Information," for more information about using queries to select target machines.

5. Optional: To modify the timeout settings for this profile, click the **Advanced Options** menu, and then select **Timeout Settings**. The Timeout Settings dialog box is displayed.



From this dialog box, you can set the following options:

Deadline The time at which a distribution expires. If a target has not received a distribution by this time, MDist 2 no longer attempts the distribution. The default value for this option is 72 hours from the current time.

Notification interval The frequency interval at which bulk data that is returned for a distribution is sent from one repeater to the next. The default value is 1 minute.

Send timeout The timeout value for data being sent from the server to the target (for example, a repeater distributing an inventory profile to an endpoint). Each time a packet is sent over the network, the

timeout for that packet is set to this value. The default is no timeout.

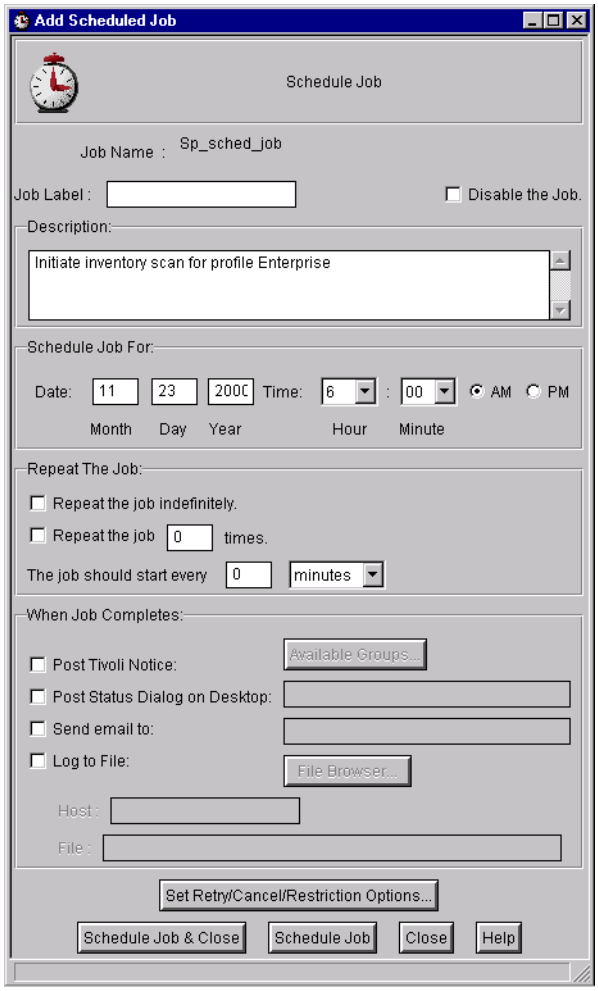
Execution timeout

The length of time a collector waits for an endpoint to return results (the downcall returns) after all of the data has been sent. This value corresponds to the *execute_timeout* tuning parameter keyword that is set using the **wmdist** command and **–s** option. (For more information about the **wmdist** command, see the *Tivoli Management Framework Reference Manual*.) If you plan to run a scan that will to take a long time to complete, for example if you must scan a large number of drives, you would set this value to allow for that time. The default is no timeout.

6. Complete one of the following options:

   - Press **Distribute & Close** to distribute the profile to the selected targets. If the scan fails for any of the targets, a dialog box is displayed to inform you which subscribers failed.
   - Press the **Schedule** button to schedule scans to automatically retry a scan to any target for which the scan might fail.

     The **Add Scheduled Job** dialog box is displayed. For more information about the Tivoli scheduler, see the *Tivoli Management Framework User's Guide*.



Command Line

To use the **wdistinv** command to distribute an inventory profile, enter the following:

```
wdistinv @InventoryConfig:First_Scan \
@Endpoint:sly @Endpoint:robbie
```

where:

@InventoryConfig:First_Scan

Specifies the name of the inventory profile.

@Endpoint:sly @Endpoint:robbie

Specifies to distribute the profile to the endpoints named **sly** and **robbie**.

Performing an Endpoint-Initiated Scan

Tivoli Inventory enables you to initiate a scan from an endpoint instead of distributing the scan from an inventory profile. This feature has many uses. For example, you can use endpoint-initiated scans to perform the following functions:

- Scan a machine each time it is rebooted.
- Scan a laptop system each time a mobile user logs in to the network.
- Scan a machine that is disconnected from the network.

You can run an endpoint-initiated scan automatically using a script or manually using the **wepscan** command. For more information about **wepscan**, see "wepscan" on page B-31.

To initiate a scan on an endpoint, you must first distribute a configuration file to the endpoint. You can create a profile that distributes a configuration file to an endpoint without scanning it by selecting the Distribute configuration file distribution option in the Global Properties window. For more information about creating an inventory profile, see Chapter 5, "Working with Inventory Profiles."

The following procedure is provided as a sample to demonstrate one use of the endpoint-initiated scan feature. This procedure starts a scan on a Windows NT endpoint each time a user logs in. This type of endpoint-initiated scan requires two tasks: setting up the endpoint environment and initiating the scan. This example combines both tasks in a batch file and then runs the batch file whenever the user logs in.

To set up this type of endpoint-initiated scan, you would need to follow these general steps:

1. Create a batch file similar to the following example:

```
@ECHO OFF
if %SYSTEMROOT%/==/ set SYSTEMROOT=%WINDIR%
if %SYSTEMROOT%/==/ goto bad
CALL %SYSTEMROOT%\TIVOLI\LCF\1\lcf_env.cmd
if %LCFROOT%/==/ goto bad
if not exist %LCFROOT%\inv\SCAN\wepscan.exe goto notfound
%LCFROOT%\inv\SCAN\wepscan
goto quit
:bad
echo Could not source environment. Scan aborted.
goto quit
:notfound
echo Could not find Tivoli Inventory Endpoint Scanner.
:quit
echo Done.
```

2. Place this batch file on your login server in a shared directory or on the local computer that you would like to scan on login.
3. Open the Windows NT User Manager. In the User Properties dialog box, access the User Environment Profile. In the Logon Script Name text box, type the full path and name for the batch file in the text box.

Populating the Configuration Repository

Tivoli Inventory saves the results of scans in the configuration repository. The configuration repository contains the tables that store the information collected during inventory profile distributions. The configuration repository schema describes the contents of the tables in the configuration repository and the relationships among the various tables. For a graphic representation of the schema, see the *Tivoli Inventory Configuration Repository* poster included with this documentation. Appendix C, “Configuration Repository,” provides a complete list of the tables, views, and queries provided with Tivoli Inventory.

When you run a hardware or software scan on target machines, Tivoli Inventory collects a pre-defined set of hardware and software components from the MIF files generated by the scanners. However, you can also use an inventory profile distribution to gather and save custom information that is not collected by a hardware or software scan. See Chapter 7, “Collecting Custom Information with Tivoli Inventory,”

for information about using custom MIF files and extending the configuration repository. See “Adding a Custom Table to the Configuration Repository” on page 7-20 for instructions and example scripts that demonstrate how to create a new table. See “Viewing User Information” on page 7-34 for instructions and example scripts that demonstrate how to create a new view.

Where to Go from Here

Once you have scanned the systems in your Tivoli environment, you are ready to review the information you have collected in the Tivoli Inventory configuration repository. See Chapter 8, “Querying Inventory Information,” for information about creating and using queries.

7

Collecting Custom Information with Tivoli Inventory

This chapter covers the procedures for collecting custom and optional information with Tivoli Inventory. You can collect custom information by creating custom software signatures. You can also add custom tables, collect custom information with UserLink, and then populate the custom tables. Finally, this chapter also describes how users can provide custom information to you.

With some additional configuration, you can extend the functionality of an inventory profile distribution. Before you do any custom configuration, you should be familiar with the options on the inventory profile. See Chapter 5, "Working with Inventory Profiles," for instructions about customizing an inventory profile.

When you customize an inventory profile, you specify how to scan target machines and what information to save. In addition to creating inventory profiles, you can use Tivoli Inventory to collect custom information with Tivoli Inventory's Web-based interface for administrators and UserLink for Tivoli Inventory, a Web-based interface for users.

The Web-based interface for administrators provides the User Data Template, a Web form that enables the administrator to specify the custom information that users provide. The User Data Template also specifies the MIF groups and attributes that the useradd.mif file used by UserLink for Tivoli Inventory will contain. The attributes that you specify with the User Data Template appear as text boxes on the User

Data Form, a Web form that users can access and complete with user information. See “Gathering User Information” on page 7-5.

The Web-based interface for users, UserLink for Tivoli Inventory, provides the User Data Form, a Web form available through UserLink that enables users to enter information about their machines that is not gathered during a hardware or software scan. When a user clicks the Submit button on this form, the useradd.mif file is created. See “Using Custom MIF Files” on page 7-36.

Working with Software Signatures

A software signature is the set of information that identifies a certain software package, such as the name and size of the executable file for the software package. The default set of software signatures provided with Tivoli Inventory was installed in the configuration repository during the installation process. See “Installing Software Signatures” on page 3-21. You can edit the software signatures provided with Tivoli Inventory, delete them, or you can add your own software signatures. For example, you can add a software signature that is not currently provided with Tivoli Inventory or for an application that was developed in-house.

Tivoli Inventory uses software signatures to determine which software packages are installed on the machines you scan. When you run a software signature scan on an endpoint, Tivoli Inventory distributes the software signatures to the endpoint, and then compares each file on the endpoint to the list of software signatures. When a file matches a software signature, the software signature data for that file is sent to the configuration repository. Because only data for matching files is sent, this scan returns less data to the configuration repository than scans for basic file information or header information.

Software signature data includes the name, size, and usually the Quick checksum value of the file used to identify the software product. This identifying file is usually the primary executable file for the product, for example notepad.exe. The software signature data collected during a scan is stored in the configuration repository in the MATCHED_SWARE table. You can use the INVENTORY_SWARE query to return all the software information about a machine.

Software signature scans return data for matching files only. Therefore, if you scan your enterprise using only software signature scans, the configuration repository contains data only for matched files, not for all files on each scanned system. After you create a new software signature or edit an existing one, you must then rescan your enterprise to gather the new or revised software signature data.

However, if you have performed a scan for basic information, the configuration repository contains information about all files on scanned systems, not just those that match software signature data. In this instance, you can write a query to view the desired file information, rather than modifying the software signatures and then rescanning your enterprise. Querying the configuration repository for basic information is similar to the functionality of Tivoli Inventory, Version 3.6.*x*.

You can use either the Signatures window or the **winvsig** command to add, edit, or delete software signatures in the configuration repository.

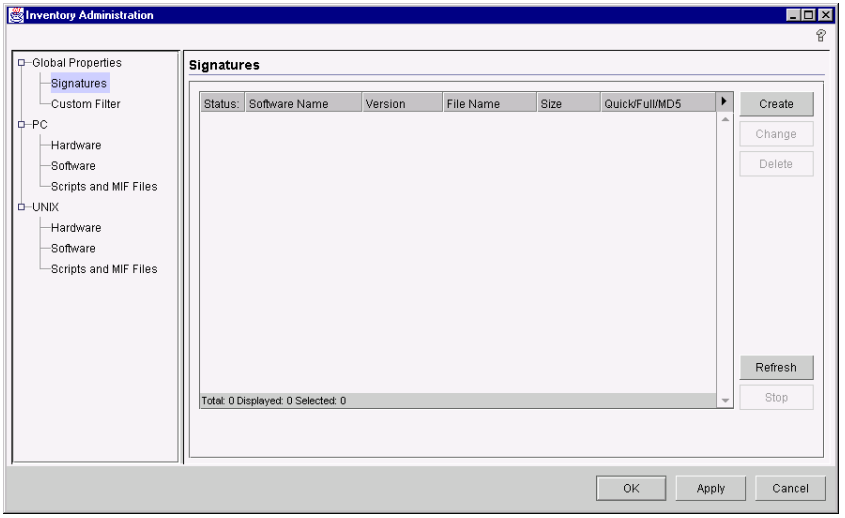
The following table provides the authorization roles required to perform this task:

Operation	Context	Role
Add, edit, or delete software signatures in the configuration repository	Signatures window	**RIM_view**, **RIM_update**, **senior**, or **super**

Desktop

To access the Signatures window, click the box next to **Global Properties** in the left pane of the Inventory Administration window to

expand the Global Properties view if necessary, and then select **Signatures**. The Signatures window is displayed.



See the Tivoli Inventory online help for the procedure to add, edit, or delete a software signature in the configuration repository using the Signatures window.

Command Line

You can use the **winvsig** command to add or delete software signatures. For example, to add a software signature for the Microsoft Office 2000 software package to the configuration repository, enter the following command:

```
winvsig –a –n MSOFFICE.EXE –s 405560 \
–d “Microsoft Office 2000” –v “9.0.2601”
```

where:

–a

Specifies that a software signature should be added to the configuration repository.

–n MSOFFICE.EXE

Specifies the name of the file used to identify the software package.

–s 405560

Specifies the size of the MSOFFICE.EXE file, in bytes.

–d "Microsoft Office 2000"

Specifies a description of the software. This corresponds to the Software Name text box in the Edit Signature Entry and New Signature Entry dialog boxes.

–v "9.0.2601"

Specifies the version of the software that the signature represents.

The following example removes a software signature for Microsoft Word 97 software from the configuration repository:

```
winvsig –r –n WINWORD.EXE –s 5317904
```

where:

–r

Specifies that a software signature should be removed from the configuration repository.

–n WINWORD.EXE

Specifies the name of the file that identifies the software signature to be deleted.

–s 5317904

Specifies the size of the WINWORD.EXE file, in bytes.

See Appendix B, "Commands," for detailed information about the **winvsig** command.

Gathering User Information

Gathering user information with Tivoli Inventory is a two-part process that requires both administrators and users to access the Tivoli Inventory Web pages. First, administrators use a collection of forms named the

User Data Template. Next, users complete the User Data Form, which is a Web form that is accessible through UserLink for Tivoli Inventory.

The User Data Template enables you to specify the text boxes on the User Data Form. When you add or delete MIF groups or attributes with the User Data Template, you are indirectly specifying which MIF groups and attributes will comprise the MIF file used by the User Data Form, useradd.mif, the next time it is generated on target machines. The groups and attributes you create are stored in the Tivoli name registry.

Each time users access UserLink for Tivoli Inventory and open the User Data Form, the current set of groups and attributes in the name registry is used to dynamically create the User Data Form Web page. A text box for each attribute is displayed on the User Data Form. (For more information about the name registry, see "Registered Names" on page B-3.)

When users click the Submit button on the User Data Form, the useradd.mif file is created and stored on the local machine. To save the information in the useradd.mif file into the configuration repository, you must customize an inventory profile to read useradd.mif during the profile distribution. You do this using the Scripts and MIF Files window in the Tivoli Inventory graphical user interface (GUI).

When the information is saved in the configuration repository, it will be saved in the tables and columns that you created to correspond with the MIF groups and attributes that comprise the useradd.mif file.

Overview

The example in the following sections illustrates how to collect the first name, last name, employee number, and office number from users and how to associate that information with the machine where it was entered. The following is an overview of the procedures required to use the Tivoli Inventory Web interface to gather custom user information.

1. Determine what information you want to gather from users and how that information should be organized in the configuration repository schema. See "Planning to Collect User Information" on page 7-7.
2. Determine which end users should have access to UserLink for Tivoli Inventory. Create one or more Tivoli administrators that

have authorization for UserLink. See “Giving Users Access to UserLink” on page 7-8.

3. Add new tables to the configuration repository to hold the custom information. (You should not store custom information in the pre-existing tables in the configuration repository.) See “Adding a Custom Table to the Configuration Repository” on page 7-20.
4. Create the corresponding MIF groups and attributes using the User Data Template Web forms. This determines what text boxes are included in the User Data Form, which is the Web form that users complete. See “Creating MIF Groups and Attributes with the User Data Template” on page 7-22.
5. Instruct users to connect to UserLink for Tivoli Inventory using the userlink.htm file. See “Using the userlink.htm File” on page 7-31. Users then complete the User Data Form. When they click the Submit button, the useradd.mif file is created. See “Completing the User Data Form” on page 7-31.
6. From the Tivoli desktop, distribute an inventory profile to retrieve the useradd.mif file. See “Retrieving and Saving User Information” on page 7-34.
7. To query the custom information, create a new view in the configuration repository. See “Viewing User Information” on page 7-34.

Planning to Collect User Information

You can use UserLink for Tivoli Inventory to collect any information that users can enter. For example, they could enter equipment information that is not collected by a hardware or software scan, such as telephone or fax machine information. However, this type of static information can easily be captured in a custom MIF file that you create manually one time. (See “Using Custom MIF Files” on page 7-36 for more information.) UserLink for Tivoli Inventory is best used to collect dynamic information (for example: the current location of a machine, including the office number, building number, and the user’s department number) because you can ask users to open and complete the User Data Form on a regular basis.

Giving Users Access to UserLink

Through UserLink for Tivoli Inventory, users can complete the User Data Form. Read this section if you want any or all of your users to complete this task.

To give users access to UserLink functions, you must designate them as Tivoli administrators on the Tivoli Management Framework and give them the **Inventory_end_user** TMR authorization role.

You can create a Tivoli administrator for each user. However, Tivoli recommends that you create a single administrator for many users by adding each user's TMR server login name in the Create Administrator dialog box. You can also create an administrator that includes a global TMR server user account, and then publish that name and password to all users who need UserLink access. (To prevent end users from modifying critical files on the TMR server, ensure that all system directories and directories containing the Tivoli installation are write-protected.)

Note: Users that have **senior**, **super**, **user**, or **admin** authorization in the TMR do not need the **Inventory_end_user** role to access UserLink.

When users click on any link in the UserLink for Tivoli Inventory main page, they must provide their TMR server login name and password. Doing this allows users to securely connect to the TMR server and access the Tivoli Inventory functions available from the Web.

Currently, both Tivoli Inventory and Tivoli Software Distribution provide some functionality through UserLink. If both applications are installed in your Tivoli environment, you can give users access to both applications in the Web interface by assigning the administrator both the **Inventory_end_user** and **SWDist_end_user** roles. However, you might want to separate UserLink for Tivoli Inventory users from UserLink for Tivoli Software Distribution users by creating separate administrators for each application. See the *Tivoli Software Distribution User's Guide* for more information about UserLink for Tivoli Software Distribution.

You can create an administrator for UserLink purposes from either the desktop or the command line. For more information about creating administrators, see the *Tivoli Management Framework User's Guide*.

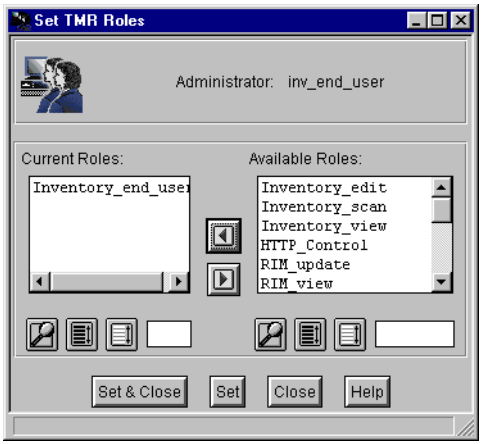
Desktop

Complete the following steps to give users access to UserLink:

1. Run the Tivoli desktop on the TMR server.
2. Double-click the **Administrators** icon to open the Administrators window.
3. Select **Create->Administrator**. The Create Administrator dialog box is displayed.

4. Complete the **Administrator Name/Icon Label** text box.
5. In the **User Login Name** text box, enter the TMR server login name of one of the users that will have access to UserLink through this administrator.
6. (Optional) Enter a group name in the **Group Name** text box. See the section about Tivoli administrators in the *Tivoli Management Framework User's Guide* for information about groups.

7. Click **Set TMR Roles**. The Set TMR Roles dialog box is displayed. Complete the following steps:

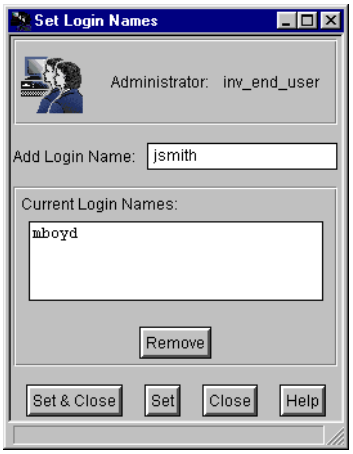


   a. Select the **Inventory_end_user** role from the **Available Roles** list. To maintain a secure Tivoli environment, do not add any other Tivoli roles.

   b. Click the left arrow button to move your selection to the **Current Roles** list.

   c. Click **Set & Close** to save your changes and return to the Create Administrator dialog.

8. Click **Set Logins**. The Set Login Names dialog box is displayed. Complete the following steps:



a. In the **Add Login Names** text box, type the TMR server login name for a user who needs access to UserLink.

b. Press the **Enter** key. The name appears in the **Current Login Names** list.

c. Repeat steps 1 and 2 above until you have added all the users that need access to UserLink.

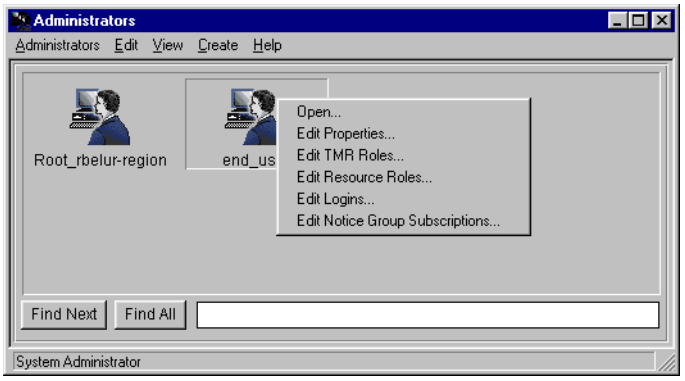
d. Click **Set & Close** to save your changes and return to the Create Administrator dialog

9. Click **Create & Close**.

The icon for the administrator you create appears in the Administrators window for your TMR.

To change settings for an administrator from the desktop, complete the following steps.

1. Right-click on the administrator icon you want to change to open the pop-up menu.



2. Select one of the following menu items to make changes to the administrator for UserLink access:

   - **Edit TMR Roles**—With this menu item you can give the administrator access to UserLink for Tivoli Inventory.
   - **Edit Logins**—With this menu item you can add, change, or delete the logins of users who are assigned to this administrator and can access UserLink.

Command Line

Use the **wcrtadmin** command to give users access to UserLink.

The following example creates the administrator **end_user_admin**. It gives **end_user_admin** access to UserLink and assigns the user login names **jbrown**, **mgonzales**, and **cwu** to the administrator:

```
wcrtadmin -l jbrown,mgonzales,cwu \
-r @distinguished:InventoryUserLink,\
Inventory_end_user end_user_admin
```

where:

–l jbrown,mgonzales,cwu
: Specifies the TMR server user IDs of the users you want to designate as administrators.

–r @distinguished:InventoryUserLink
: Specifies the distinguished object that is created during the installation of Tivoli Inventory to hold the methods needed for UserLink operations.

Inventory_end_user
: Specifies the Tivoli role assigned to the **end_user_admin** administrator.

end_user_admin
: Specifies the label for the administrator icon on the Tivoli desktop.

You can use the **wsetadmin** command to add more logins at a later time. The **wsetadmin** command also provides you with a way to give an existing administrator authorization for UserLink for Tivoli Inventory. The following example gives the administrator **end_user_admin** access to UserLink for Tivoli Inventory.

```
wsetadmin –r @distinguished:InventoryUserLink,Inventory_end_user \
end_user_admin
```

where:

–r @distinguished:InventoryUserLink
: Specifies the distinguished object that is created during the installation of Tivoli Inventory to hold the methods needed for UserLink operations.

Inventory_end_user
: Specifies the new authorization role assigned to the administrator.

end_user_admin
: Specifies the administrator whose settings are being changed.

See the *Tivoli Management Framework Reference Manual* for information about **wcrtadmin** and **wsetadmin** command syntax and usage. Rather than entering each new login name individually through the Tivoli desktop or with the **wsetadmin** command, you might want to create a script to automate this process.

Designing Custom Tables

After you determine what information you want to collect from users, decide how that information will be organized in the configuration repository schema. The pre-defined tables in the database schema are to be populated only with the results of Tivoli Inventory hardware and software scans. For this reason, to use the useradd.mif file to gather information from users, you must create tables and columns in the configuration repository to store the information.

When you know what information you want users to provide, design the tables and columns that you will add to the configuration repository to store the information. Consider the following:

- The number of new tables and the number of columns in each table. Consider how much information you want to collect and how often to collect it.
- The names for the new tables and columns. When selecting names for the new tables and columns consider the following:
  - The table and column names must be the names of the useradd.mif file groups and attributes, respectively. For example, if you name the new table USER_INFO, you must also use USER_INFO for the name of the MIF group. The information stored in the USER_INFO group in the useradd.mif file is stored in the USER_INFO table. Similarly, the column names you select for the USER_INFO table are the names that you must use for the MIF attributes that are included in the USER_INFO MIF group.
  - DB2 and Informix table and column names are limited to a maximum of 18 characters. Refer to “Naming Custom Tables, Views, and Columns for DB2 and Informix” on page 7-16 for a list of truncated names that you can use as a guide when naming tables and columns for a DB2 database. Moreover, if you plan to create a history table for your custom table, you must limit the name of the custom table to a maximum of 16 characters. See “Creating History Tables for Custom Tables” on page 7-19 for more information.

- You can name tables and columns using the following characters:
  - Lowercase a through z
  - Uppercase A through Z
  - Numeric characters 0 through 9
  - Underscores

  You cannot use spaces; table and column names must start with a letter.
- Custom tables and columns should not have the same name as existing tables and columns in the configuration repository. See Appendix C, "Configuration Repository," or the *Tivoli Inventory Configuration Repository* poster for existing table and column names.
- Do not use SQL-92 reserved words for custom table and column names. For more information about SQL-92 reserved words, consult a database administrator or SQL documentation.

■ To determine the information type of each column, consider what type of information is stored in each column. The column types you use must match the attribute types you select when adding attributes with the User Data Template.

■ To determine the location of the tables in the configuration repository schema, consider how the information should be organized in relation to other Tivoli Inventory information.

■ Consider whether the new tables should be related to existing tables in the schema. You can associate the tables you add to existing tables in the schema by using primary and foreign keys. For example, to relate a new table to an existing table in the configuration repository, make the primary key of the existing table a foreign key in the new table.

■ The column COMPUTER_SYS_ID must be included in the primary key of each custom table.

Naming Custom Tables, Views, and Columns for DB2 and Informix

DB2 and Informix restrict database table, view, and column names to 18 characters. Because of this limitation, the RDBMS Interface Module (RIM) truncates the table, view, and column names described in the following table. When you create custom tables, views, or columns in the configuration repository, use the truncated name rather than the full name.

Standard Name	Truncated Name
ACTIVATION	ACTIV
ADDRESS	ADDR
ADMINISTRATOR	ADMIN
ARCHITECTURE	ARCH
AVAILABLE	AVAIL
BLOCK	BLK
BRIDGE	BRG
CHANGE	CHG
COMPONENT	COMP
CONFIG	CFG
CONNECTIONS	CONX
CONSOLE	CONS
CONTROLLER	CNTRL
CONVENTIONAL	CONV

Standard Name	Truncated Name
COPROCESSOR	COPROC
DESCRIPTION	DESCRIP
DEVICE	DEV
DIRECTORY	DIR
DISTRIBUTION	DIST
DRIVERS	DRVS
DRIVER	DRV
FAXMODEM	FMODEM
FILEPACK	FPACK
FILESYSTEM	FILESYS
FLOPPYDRIVE	FLPY_DRV
HARDDISK	HDISK
HARDWARE	HWARE
HISTORY	HIST
INSTALLED	INST
INSTALLATION	INSTL
INTERNET	INET
INTERRUPT	INT
KERNEL	KRNL
LANGUAGE	LANG

Standard Name	Truncated Name
LASTSCAN	LSCAN
LOGICALDRIVE	LOG_DRV
MEMORY	MEM
NETWARE	NWARE
NETWORK	NET
NUMBER	NUM
PLATFORM	PLTFRM
PROFILE	PRFL
RELATIONSHIP	REL
RESOLUTION	RES
RESTRICT	RSTRT
RWOPTICAL	RWOPT
SECURITY	SEC
SERIAL	SER
SERIALNUMBER	SERNUM
SERVICE	SVC
SIGNATURE	SIG
SOFTWARE	SW
START	ST
SUPPORT	SUPP

Standard Name	Truncated Name
SYSTEM	SYS
SYSTEMS	SYS
VERSION	VER
VIDEO	VID
VIRTUAL	VIRT
VOLUME	VOL
VOLUMELABEL	VOLLABEL
DRIVE	DRV

Creating History Tables for Custom Tables

To use history tracking with custom tables, you must create a history table when you create your custom table. You must name the history table the same name as the custom table, prepended with an uppercase H and an underscore character (H_). For example, if your custom table is named MY_TABLE, the history table for this custom table must be named H_MY_TABLE.

If you are using a DB2 or Informix RDBMS for your configuration repository, you must restrict the names of your history tables to a maximum of 18 characters, including the H_ prefix, to comply with the naming convention for history tables.

The history table must contain the same column names as the custom table. Each column must be the same data type as the corresponding

column in the custom table. In addition, the history table must contain the following columns:

Column Name	Data Type
RECORD_ACTION	CHAR(6)
PRFL_ACTION	VARCHAR(20)
RECORD_TIME	TIMESTAMP, or appropriate date structure for your RDBMS.

Tivoli Inventory does not use constraints in history tables. If you create a custom history table correctly, Tivoli Inventory automatically detects and populates the history table.

Adding a Custom Table to the Configuration Repository

You must add the new tables and columns to the Tivoli Inventory configuration repository before you use Tivoli Inventory to collect user information. When you add a table, you specify the table name, number of columns in the table, data type and width of each column, and whether any of the columns are a primary key, foreign key, or both. Before you create a new table, you should be familiar with SQL and relational database concepts, such as primary keys, foreign keys, data types, NULL fields, and so on.

When adding a custom table, watch for any error messages that indicate the table was not created correctly.

The following SQL example adds the USER_INFO table to the configuration repository in an Oracle RDBMS. The USER_INFO table can be updated when information changes and includes the following columns:

- LAST_NAME
- FIRST_NAME
- EMP_NUMBER

- OFFICE_NUMBER
- COMPUTER_SYS_ID

In this table, when users enter their names, employee numbers, and office numbers into the User Data Form, the information will be associated in the configuration repository with the computer system ID (COMPUTER_SYS_ID) of their machine.

The useradd.mif file created to match this table includes a group named USER_INFO that consists of attributes named LAST_NAME, FIRST_NAME, EMP_NUMBER, and OFFICE_NUMBER.

Note: The following example is for an Oracle configuration repository.

To add the USER_INFO table, first log into the configuration repository as the configuration repository user or owner. Next, type the following from the command line:

```
DROP TABLE USER_INFO;

CREATE TABLE USER_INFO(
   COMPUTER_SYS_ID      varchar2(64) NOT NULL,
   LAST_NAME            varchar2(64) NOT NULL,
   FIRST_NAME           varchar2(64) NOT NULL,
   EMP_NUMBER           varchar2(32) NOT NULL,
   OFFICE_NUMBER        varchar2(12) NOT NULL,
constraint USERINFO_PK PRIMARY KEY (COMPUTER_SYS_ID),
constraint USERINFO_FK FOREIGN KEY (COMPUTER_SYS_ID)
references COMPUTER (COMPUTER_SYS_ID)
on delete cascade
);
commit;
```

The foreign key constraint in this script is necessary if you want to use the **winvrmnode** command to remove all information about an endpoint from this table when the endpoint is deleted. This constraint enables you to perform a *cascading delete*: when you delete a value from the COMPUTER table that is referenced by a foreign key in this table, the value associated with the foreign key in this table is also deleted. For more information about cascading deletes, consult a qualified database administrator. For examples of cascading deletes and their corresponding triggers that are appropriate to your RDBMS, see the schema installation scripts in $BINDIR/../generic/inv/SCRIPTS/RDBMS.

If you want to create a history table for the USER_INFO table, type the following from the command line:

```
DROP TABLE H_USER_INFO;

CREATE TABLE H_USER_INFO(
   COMPUTER_SYS_ID      varchar2(64),
   LAST_NAME            varchar2(64),
   FIRST_NAME           varchar2(64),
   EMP_NUMBER           varchar2(32),
   OFFICE_NUMBER        varchar2(12),
   RECORD_ACTION        char(6),
   PRFL_ACTION          varchar2(20),
   RECORD_TIME          DATE
);
commit;
```

Note that no constraints are used for the history table.

If you want to query the custom information, you might also want to create a new view. See "Viewing User Information" on page 7-34 for more information. You can also use the **wcrtquery** command to add custom queries to your query library. See the *Tivoli Management Framework Reference Manual* for more information.

Creating MIF Groups and Attributes with the User Data Template

After you add the custom table or tables needed to store user information in the configuration repository, the next step is to create the corresponding MIF groups and attributes with the User Data Template, a set of HTML forms provided with Tivoli Inventory. Use the names of the custom tables as the names of the MIF groups and the names of the columns as the names of the attributes in the User Data Template. Create a group for each new table you added to the configuration repository, and use the table name as the name of the group. The group name must match the table name exactly, including case. For example, the MIF group that corresponds with the USER_INFO table also must be named USER_INFO. Create attributes within the group for each of the columns in the table. For example, the USER_INFO group consists of attributes named LAST_NAME, FIRST_NAME, EMP_NUMBER, and OFFICE_NUMBER, which match the names of the columns in the table USER_INFO.

If you want to change the information that you collect from users, you can use the User Data Template to delete MIF groups or attributes. To delete a MIF group, you must first delete all the attributes in the group. When you delete a MIF attribute, the text box that corresponds to that attribute is not displayed on the User Data Form the next time users access the form, and the column in the configuration repository by that name is not updated. You can also add attributes to collect more information, but, before you retrieve the information, you must add the corresponding columns to the tables in the configuration repository.

To create the USER_INFO group and its attributes with the Tivoli Inventory Web pages for the administrator, complete the following steps:

1. To access the User Data Template Web page, open the following URL with a Web browser:

   http://*ServerName:OservPortId*

   where *ServerName* is the host name of the TMR server, and *OservPortId* is the number of the port where the oserv is listening. By default, the port number for the oserv process is 94. An example of this URL might be **http://ball:94**.

   The TME 10 Web Access page is displayed.

2. Click the **Tivoli Inventory** link. The Web browser displays a dialog with user name and password text boxes. (If there is not a Tivoli Inventory link, Tivoli Inventory has not been installed.)

3. Enter your Tivoli administrator login name and the password you used to log in to the machine. If authentication fails, ensure that the following items are set correctly:

   - When you run the **odadmin** command, "Remote client login allowed" should be set to TRUE. If it is not, enter the following command:

     ```
     odadmin set_allow_rconnect TRUE
     ```

   - Your administrator roles must include **RIM_update** and **RIM_view**.

   - On a TMR server running a UNIX operating system, the user and password must be included in the **/etc/passwd** file as valid login IDs.

- If the TMR server is an NT primary domain controller (PDC) or a backup domain controller (BDC), enter *Domain\Login* in the **User Name** field, where *Domain* is the name of the domain controlled by the PDC or BDC and Login is the administrator's login name.

  Note: Once you are authenticated, the Tivoli Inventory Web pages can be accessed until the browser is closed.

4. To create MIF groups and attributes, click the **User Data Template** link. The Tivoli Inventory User Data Template is displayed.

5. To add a MIF group named USER_INFO, complete the following steps:

   a. Click the **Add MIF Group** link. The Add MIF Group page is displayed.

Add MIF Group
Current MIF Groups
MIF Group to be Added
Enter a name and description for the MIF group to be added.
The group name will also be the name of a table in the configuration repository. Therefore, use uppercase letters and do not use spaces in the name.
Group Name
Group Description
Add
Reset
Return to User Data Template

b. In the Group Name text box, enter a name for the MIF group, for example, **USER_INFO**. Enter the name in uppercase. Do not include spaces, because the name must also be the name of a table in the configuration repository. Spaces are not allowed in table names.

c. In the Group Description text box, enter a description of the MIF group, for example, **Employee Information**. A description is required by the Desktop Management Interface (DMI) 2.0 standard; it is displayed on the User Data Form.

d. Click **Add** to add the MIF group. A Results page is displayed.

6. To add an attribute to the USER_INFO group, click the **Return to User Data Template** link. The User Data Template is displayed.

   a. Click the **Add or Delete MIF Attribute** link.

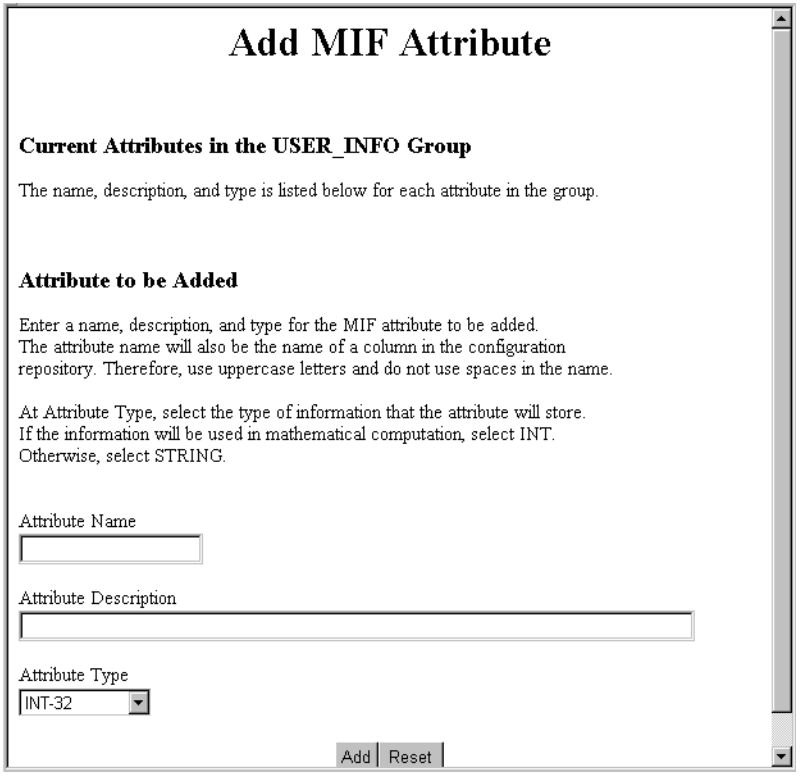
      The Add or Delete MIF Attribute page is displayed.

Add or Delete MIF Attribute
Current MIF Groups
Select the MIF group to which the attribute will be added or deleted.
USER_INFO - Employee information
Operation
Add Attribute
Delete Attribute
Go
Return to User Data Template

b. Select the MIF group to which the attribute is to be added.

c. Select **Add Attribute** and click **Go**.

The Add MIF Attribute page is displayed.


Add MIF Attribute
Current Attributes in the USER_INFO Group
The name, description, and type is listed below for each attribute in the group.
Attribute to be Added
Enter a name, description, and type for the MIF attribute to be added.
The attribute name will also be the name of a column in the configuration repository. Therefore, use uppercase letters and do not use spaces in the name.
At Attribute Type, select the type of information that the attribute will store.
If the information will be used in mathematical computation, select INT.
Otherwise, select STRING.
Attribute Name
Attribute Description
Attribute Type
INT-32
Add
Reset

d. In the Attribute Name text box, enter a name for the attribute, for example **LAST_NAME**. Do not use spaces in the attribute name, because this is also be the name of a column in the configuration repository. Spaces are not allowed in the Tivoli Inventory configuration repository table or column names.

e. In the Attribute Description text box, enter a description for the attribute, for example **Last Name**. This description is displayed on the User Data Form next to a text box.

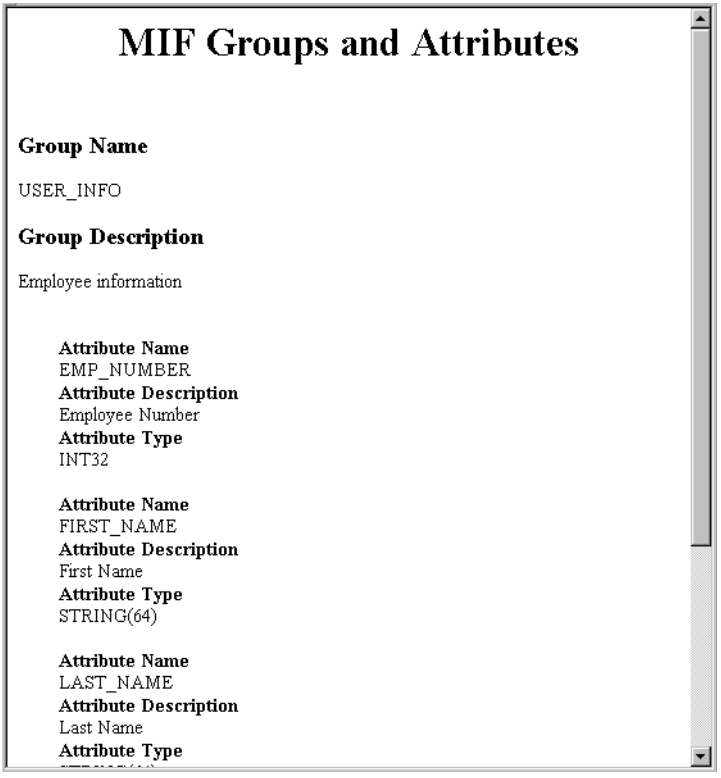
f. From the Attribute Type list, select a type. Select **INT** if the information will be used in a mathematical computation. Otherwise, select **STRING**. The data type you select must

correspond with the data type you specified for the corresponding column. For this example, select **STRING-64**, because this corresponds to the data type and length of the column named LAST_NAME in the USER_INFO table that was added in "Adding a Custom Table to the Configuration Repository" on page 7-20.

g. Click **Add** to add the attribute to the MIF group. A Results page is displayed.

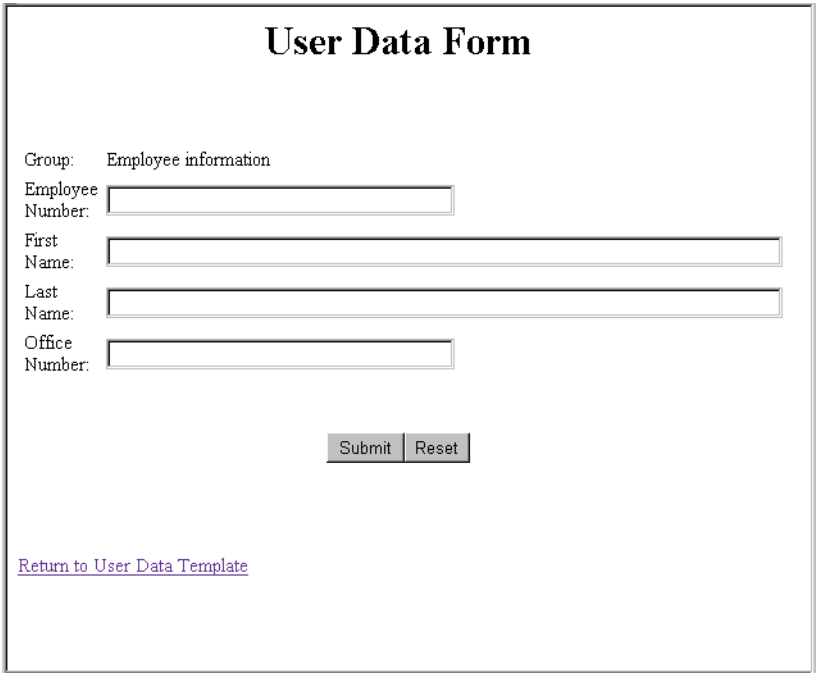
h. Repeat steps a through g for every MIF group and attribute you want to include in the useradd.mif file.

7. To see a current list of MIF groups and attributes, click the **View MIF Groups and Attributes** link. The MIF Groups and Attributes page is displayed.



8. To see how the information you have entered is displayed on the User Data Form, click the **Preview User Data Form** link on the User Data Template page.

   A preview of the User Data Form that users will see is displayed.



Using the userlink.htm File

To connect to the UserLink for Tivoli Inventory features, users open an HTML file named userlink.htm with a Web browser. Endpoints receive the userlink.htm file when they log into the gateway. The file is stored in the $DRIVE:/etc/Tivoli/$LANG directory, where $DRIVE is the drive where the Tivoli agent is installed and $LANG is the language used on the endpoint. For more information about $LANG, see the *Tivoli Management Framework Release Notes*.

Note: It is recommended that users add the userlink.htm page to the list of bookmarks in their Web browsers so they can access it easily.

Completing the User Data Form

You must instruct users to access UserLink for Tivoli Inventory and complete the User Data Form with the relevant information. The User

Data Form is generated each time users click the User Data Form link from UserLink. To ensure that this form includes the most current text boxes, instruct users to reload the User Data Form before entering data into it.

When the users complete the User Data Form, the useradd.mif file is created on the endpoint in the $LCF_BASE_DIR/inv/SCAN directory. The $LCF_BASE_DIR is the directory in which you installed the Tivoli agent and $oid is the Tivoli object ID of the client.

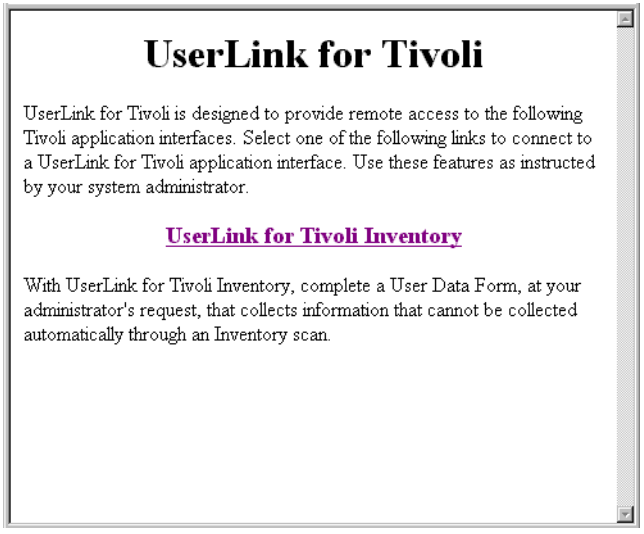
It is recommended that you either immediately retrieve and save the useradd.mif file in the configuration repository or move it to a different directory after it is generated. The useradd.mif file is overwritten every time a user clicks the **Submit** button on the User Data Form..

To connect to UserLink and complete the User Data Form, follow these steps:

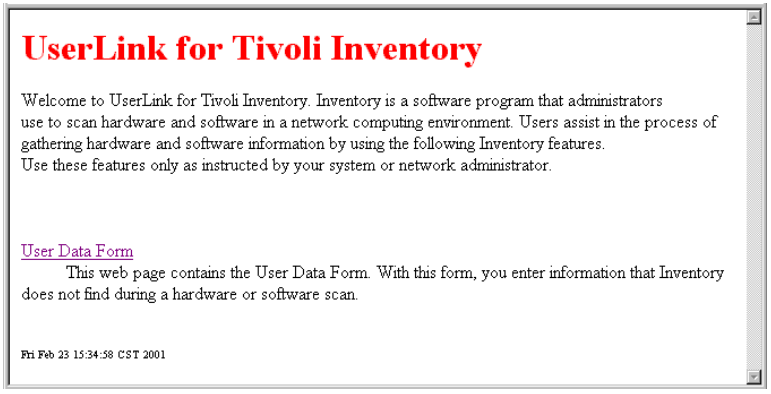
1. Open the userlink.htm file with a Web browser. See “Using the userlink.htm File” on page 7-31 for location information.

   The preliminary UserLink for Tivoli page is displayed.

2. Click the **UserLink for Tivoli** link. The interior UserLink for Tivoli page is displayed.

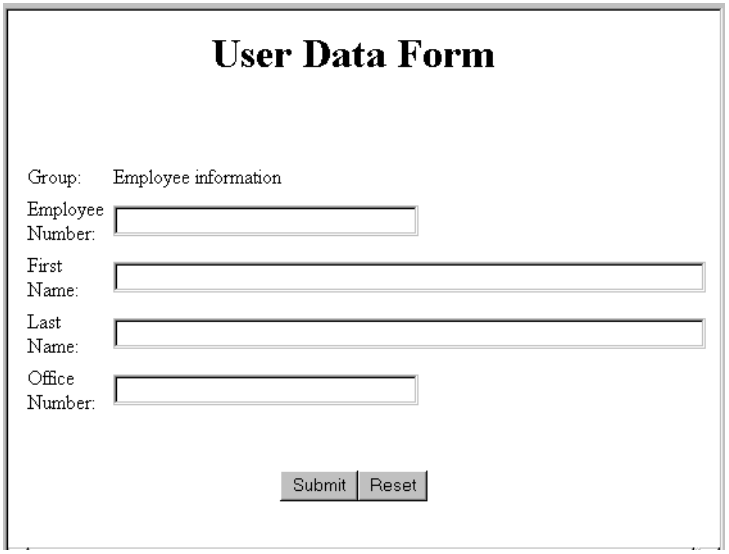

UserLink for Tivoli
UserLink for Tivoli is designed to provide remote access to the following Tivoli application interfaces. Select one of the following links to connect to a UserLink for Tivoli application interface. Use these features as instructed by your system administrator.
UserLink for Tivoli Inventory
With UserLink for Tivoli Inventory, complete a User Data Form, at your administrator's request, that collects information that cannot be collected automatically through an Inventory scan.

3. Click the **UserLink for Tivoli Inventory** link. The UserLink for Tivoli Inventory page is displayed.


UserLink for Tivoli Inventory
Welcome to UserLink for Tivoli Inventory. Inventory is a software program that administrators use to scan hardware and software in a network computing environment. Users assist in the process of gathering hardware and software information by using the following Inventory features.
Use these features only as instructed by your system or network administrator.
User Data Form
This web page contains the User Data Form. With this form, you enter information that Inventory does not find during a hardware or software scan.
Fri Feb 23 15:34:58 CST 2001

4. Click the **User Data Form** link. A prompt for your user name and password is displayed.

5. Enter the name and password configured for the end user administrator. The User Data Form page is displayed.



6. Complete the text box or text boxes on the User Data Form with the appropriate information.

7. Click **Submit**. (When the users complete the User Data Form, the useradd.mif file is created on the local machine.)

Retrieving and Saving User Information

After all the users have completed the User Data Form on their machines, you can collect the useradd.mif file from their machines and save the user information into the configuration repository. The useradd.mif file is the MIF file that contains the information that the user entered. During the distribution, the information in the useradd.mif file is saved in the configuration repository in the custom tables you created. To retrieve the useradd.mif file, distribute an inventory profile that has been customized to read the useradd.mif file. To customize the inventory profile, enter useradd.mif, including its path, in the text box for custom MIF files in the Scripts and MIF Files window.

Viewing User Information

RDBMS views are created so groups of information in a database can be accessed easily by a query. A view is a way to group information from related tables. For example, the PROCESSOR_ MODEL, PROCESSOR_SPEED, and COMPUTER_SYS_ID columns reside in different tables throughout the configuration repository. A view named PROCESSOR_VIEW includes all of the information in these columns. So, instead of running a query for each table, you can access all the information with one query by using PROCESSOR_VIEW.

The views that are provided with Tivoli Inventory do not include the custom tables or columns added to the configuration repository. To query custom information, create a new view by creating a script which adds the new view to the configuration repository.

Remember the following rules when creating a new view:

- Do not edit the views provided with Tivoli Inventory.
- Make sure that the new view contains the TME_OBJECT_ID and TME_OBJECT_LABEL columns. If you want to create a subscription list using the **wruninvquery** command, run one of the subscription queries provided with Tivoli Inventory or run a new query that returns the names of machines that meet the query criteria.

- Add the name of the view to the QUERY_VIEWS table in the configuration repository. If you do not add the name of the view to this table, the view will not be available from the ellipses (**...**) button on the Create Query or Edit Query dialog. See "Creating a Query" on page 8-2 for more information about selecting views from the query dialogs.

The following example creates a view named LOCATION_USER_VIEW in an Oracle configuration repository. This view enables you to query the information in the LAST_NAME, FIRST_NAME, EMP_NUMBER, and OFFICE_NUMBER columns in the custom table USER_INFO, which was created by the example in the "Adding a Custom Table to the Configuration Repository" on page 7-20. Note that some of the lines in the **select** statement consist of a table name followed by a column name, separated by a period. If you specify a column that resides in more than one of the listed tables, you must enter the table name and a period before the column name in the **select** statement.

Note: The following example applies to an Oracle RDBMS.

```
drop view LOCATION_USER_VIEW
create view LOCATION_USER_VIEW
as
select
TME_OBJECT_ID,
TME_OBJECT_LABEL,
LAST_NAME,
FIRST_NAME,
EMP_NUMBER,
OFFICE_NUMBER,
USER_INFO.COMPUTER_SYS_ID
from USER_INFO, COMPUTER
where
        USER_INFO.COMPUTER_SYS_ID =
             COMPUTER.COMPUTER_SYS_ID;

commit;
insert into QUERY_VIEWS values ('LOCATION_USER_VIEW');
commit;
```

After you write the file, save it as *filename***.sql**, and then close it. You can add the new view to the configuration repository by running the file as the user invtiv with an interactive SQL tool.

You can use the new view with any new or existing query by either entering the name of the view in the **Table/View Name** text box on the Create Query, Edit Query, and Inventory Query dialog boxes or by clicking the ellipses (**…**) button next to the **Table/View Name** text box (if you added the name of the view as a row to the **QUERY_VIEWS** table in the configuration repository).

Using Custom MIF Files

An inventory profile distribution can read any MIF file that adheres to the Desktop Management Task Force (DMTF) 2.0 MIF format. Creating DMTF-compliant MIF files enables you to use Tivoli Inventory to collect information that is not generated during a software or hardware scan. To collect custom MIF files, you set an inventory profile to read the MIF files during a profile distribution. A custom MIF file might be one that is generated by another program or a MIF file that you create manually. Tivoli Inventory generates hardware and software MIF files during a hardware or software scan, respectively. The information in these MIF files is stored in the pre-defined tables in the configuration repository. If you plan to use custom MIF files, you must create the tables and columns in the configuration repository to store the custom information.

To ensure that Tivoli Inventory can read the MIF file, use the following guidelines:

- The MIF file should be compliant with DMTF 2.0 MIF standards, including the following:
  - Each group and attribute must have a name and description.
  - The type and size must be specified for each attribute.
- Read and follow the guidelines about custom tables in the "Designing Custom Tables" on page 7-14 and "Adding a Custom Table to the Configuration Repository" on page 7-20.
- Do not modify existing tables in the configuration repository.
- MIF groups should not have the same name as existing table names in the configuration repository.
- If you use tables in the MIF file to store multiple entries within a group, use the unique attribute or attributes as the key. Then,

include those attributes as part of the primary key in the table that you create for the MIF group.

- Ensure that the name of the custom MIF file is unique within the TMR. For example, do not use the names of the MIF files that Tivoli Inventory generates during a scan. Tivoli Inventory generates the following MIF files:
  - dmiscan.mif
  - mrmbios.mif
  - tivfscan.mif
  - tivhscan.mif
  - tivrscan.mif
  - tivsscan.mif
  - tivwscan.mif
  - useradd.mif

The following is an example of how to add a new group and attribute to a MIF file.

```
START COMPONENT
NAME = "CUSTOMDATA.MIF"
DESCRIPTION = "Custom data that is not scanned"
   START GROUP
      NAME = "SYS_INFO"
      ID = 1
      START ATTRIBUTE
         NAME = "SERIAL_NUMBER"
         ID = 1
         ACCESS = READ-ONLY
         TYPE = STRING(32)
         VALUE = "123456789012"
      END ATTRIBUTE
      START ATTRIBUTE
         NAME = "MAKER"
         ID = 2
         ACCESS = READ-ONLY
         TYPE = STRING(32)
         VALUE = "Compaq"
      END ATTRIBUTE
   END GROUP

END COMPONENT
```

Customizing PC Scans for BIOS Information

You can configure Tivoli Inventory to scan PC systems for BIOS (Basic Input/Output System) information, including the date of the BIOS chip, the BIOS identification bytes, the unique string, and the manufacturer and model of the machine that is scanned. The table PC_SYS_PARAMS stores this information in your Tivoli Inventory configuration repository.

The BIOS scanner includes a dictionary that contains a number of pre-defined machine manufacturers and models. To find a machine manufacturer or model that is not in the dictionary, use the mrmbios utility to add the appropriate new information to the file bios.ini.

To update the bios.ini file, complete the following steps:

1. If you have not already done so, scan all of the PCs in your environment for BIOS information.
2. Determine if there are PCs in your network that have unknown manufacturers or models by doing the following:
   a. Query the PC_SYS_PARAMS table or the PC_BIOS_VIEW view.
   b. Look at the BIOS_MODEL column to determine if the model for a machine is not recognized. For any unrecognized models, the value of this column is UNKNOWN.
   c. Find the COMPUTER_SYS_ID column for any unrecognized model numbers to determine which systems have the unknown manufacturer or model.
3. To update the bios.ini file, run the mrmbios utility from the command line of each type of PC that has an unrecognized model (each type of PC that you identified in step 2). Enter the following:

   ```
   mrmbios -d bios.ini -a
   ```

   The following prompt is displayed:

   ```
   Enter the manufacturer of this machine:
   ```
4. Enter the name of the manufacturer and press the Enter key. The following prompt is displayed:

   ```
   Enter the model of this machine.
   ```

5. Enter the model of the machine and press the Enter key. The bios.ini file is created.
6. Merge all updated bios.ini files into one file using a text editor.
7. Save the updated bios.ini file in the $DBDIR/inventory/INI directory of the TMR server in the TMR from which you originate inventory scans.
8. Run a hardware scan on all PCs in your network to update the PC_SYS_PARAMS table in your configuration repository, based on the new bios.ini file.

Where to Go from Here

After you have learned about options to customize inventory scans and software signatures, go to Chapter 6, “Distributing Inventory Profiles,” for information about collecting inventory hardware, software, and custom information.

8

Querying Inventory Information

This chapter describes the tasks necessary to search for information in the configuration repository. The Tivoli Management Framework query facility enables you to create SQL statements to retrieve information gathered from inventory profile distributions.

The query feature consists of query libraries and queries. Query libraries reside in policy regions and are created to contain queries. Each query is designed to retrieve a specific set of information from a specific view or table in the configuration repository. There are several pre-defined queries provided with Tivoli Inventory. You can also create your own queries. This chapter describes the different ways you can use queries to get inventory information from the configuration repository, including the following:

- An explanation of how to use the pre-defined queries provided with Tivoli Inventory. (See "Using the Inventory and Subscription Queries" on page 8-2.)
- An example of how to create your own query from the Create Query dialog box. (See "Creating a Query" on page 8-2.)
- An explanation of how to run a query you have created and how to view or save the results. (See "Running a Query" on page 8-10.)
- Steps for using a query to select target machines for a Tivoli Inventory profile distribution. (See "Defining Subscribers with Queries" on page 8-14.)

- Steps for using a query to retrieve inventory information about one machine in a TMR. (See "Querying for Information about One Client" on page 8-16.)
- An explanation of how to query interconnected TMRs and display the data for each TMR separately. (See "Using Queries in Interconnected TMRs" on page 8-20.)

For a complete description of the Tivoli Management Framework query facility and query commands, see the *Tivoli Management Framework User's Guide*.

Using the Inventory and Subscription Queries

Tivoli Inventory provides two sets of queries that access Tivoli Inventory results in the configuration repository. The inventory queries retrieve different sets of hardware and software information from the configuration repository. For example, the INVENTORY_HWARE query returns basic hardware information about all machines.

The subscription queries that are provided with Tivoli Inventory return lists of machines that meet the specifications of the query. For example, the AIX_SUBSCRIPTION query returns a list of machines that have an AIX operating system.

The query libraries and queries that are provided with Tivoli Inventory are created during the installation process. See "Installing Tivoli Inventory and Subscription Queries" on page 3-22 for more information about running the inventory_query.sh, subscription_query.sh, h_inventory_query.sh, and the h_subscription_query.sh scripts to create the inventory and subscription queries.

Creating a Query

You can create and customize a query to retrieve the information you want from the configuration repository.

When you create a query, you specify the repository in which to search for information and the set of information you want to retrieve. Before you create a query, decide what information you want to access. Then determine which tables and columns in the configuration repository contain that information and include them in the query. The *Tivoli*

Inventory Configuration Repository poster shows the configuration repository data model. Appendix C, "Configuration Repository," contains a brief description of each table.

Once you know which tables and columns you need to query, you can select the view that contains those columns. See Appendix C, "Configuration Repository," for a list of the views. If there is not a view that contains all the columns that you would like to include in your query, you can create a new view. See "Viewing User Information" on page 7-34 for instructions about creating a new view.

When you create a query, you must specify the following in the Create Query dialog box:

- A **Query Name** that is unique in the TMR.
- A **Repository** that determines which tables and views you can use for the query.
- A **Table/View Name** within the repository to run the query against. The table or view that you select determines which columns you can use for the query.
- A set of **Chosen Columns** within the table or view that are part of the retrieved information.

You can also specify a description and one or more where clauses for the query. Use the Where Clause section of the dialog box to construct an SQL search clause that specifies which information the query will return. The query example in this section demonstrates how to use several features of the query facility.

For a complete description of how to use the Tivoli Management Framework query facility, see the *Tivoli Management Framework User's Guide*.

The following table provides the authorization roles required to perform this task:

Operation	Context	Role
Create a query	Query library	**Query_edit**, **admin** or **senior** or **super**

You can create this query from either the Tivoli desktop or the command line.

Desktop

The following procedure provides an example of how to create a query from the Tivoli desktop. The query that is created in this example retrieves the following information from the configuration repository:

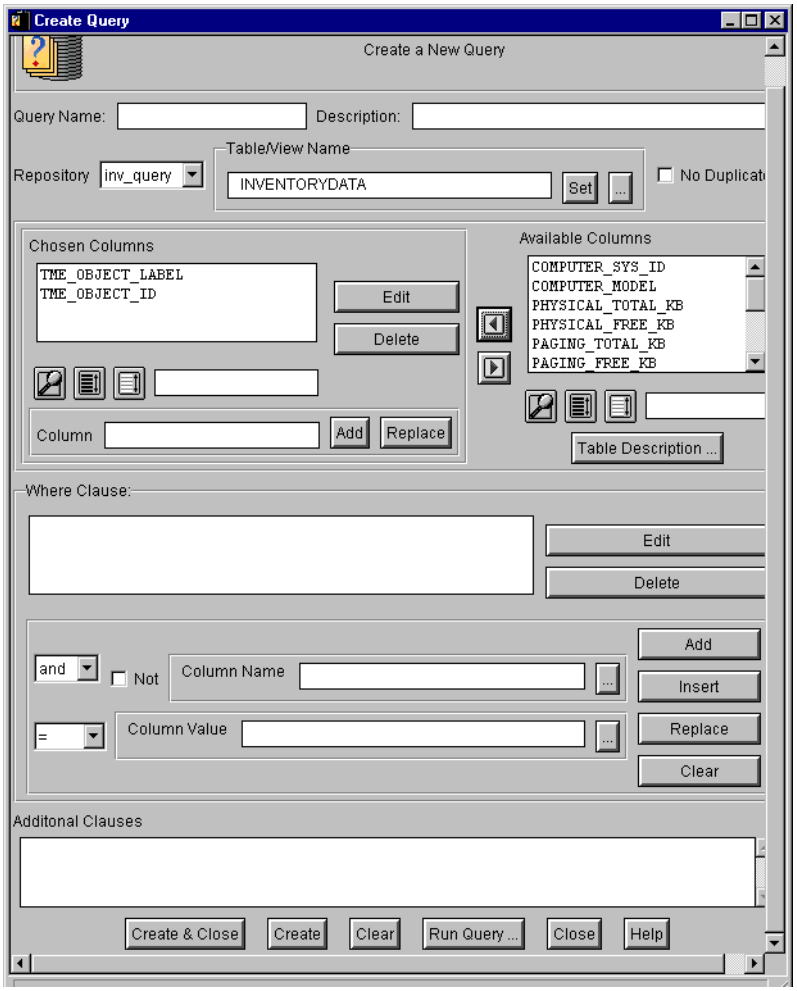
- The machines that have Microsoft Word installed and at least 32 MB of RAM available;
- The version of Microsoft Word that is installed on each machine.

Use the following steps to create a query from the Tivoli desktop:

1. From the policy region, double-click the icon of the query library in which the query will reside. The Query Library window is displayed.

   Note: Queries must be created in query libraries. You can either create a query library or select an existing query library in which to store it.

2. Select **Query** from the Create menu. The Create Query dialog box is displayed.



3. In the Query Name field, enter a unique name for the query, such as MSWord_Finder. The name can be any set of alphanumeric characters, uppercase letters, or lowercase letters. Spaces are allowed.
4. In the Description field, enter a brief description of the query. This field is optional. The description of this query will be Finds version of MSWord.

5. From the Repository menu, select an RDBMS database against which the query will run. The Repository menu shows the names of RIM objects in the local TMR. This query will access scan results, so select **inv_query** to access the Tivoli Inventory configuration repository.
6. In the Table/View Name field, you have the following options:
   - Enter the name of the view INVENTORYDATA and click the **Set** button.

     Note: You can choose more than one table or view in this field. Separate each table or view name by a comma (,). In the Available Columns list, each table or view name selected precedes the name of the column.
   - Click the ellipses (**…**) button to select from a list of views in the repository. This list shows the views listed in the QUERY_VIEWS table in the configuration repository. The views for this query are INVENTORYDATA and INST_SWARE_VIEW, because they include the columns that contain information about software and physical memory.

   Note: Changing the entry in the Table/View Name field populates the Available Columns list and clears the rest of the fields on the dialog box, including the Chosen Columns list, Where Clause, and Additional Clause fields.
7. Select **No Duplicates** to remove duplicate results from the query results.
8. In the Available Columns list, select the columns from which you want to retrieve information and move them to the Chosen Columns list with the left arrow button.

   Note: You might not be able to see the full name of the column in this list. You can expand the Available Columns list window by clicking and dragging the edge.

   Because you want this query to search for information about software and memory, choose the following columns:

- INST_SWARE_VIEW.SWARE_DESC
- INST_SWARE_VIEW.SWARE_VERS
- INVENTORYDATA.PHYSICAL_TOTAL_KB

Because the query returns a list of machines, include the TME_OBJECT_LABEL column as well so that you can identify all the machines in the query.

9. Create an SQL search clause in the Where Clause panel of the dialog box to specify what information the query will return. Use the Column Name and Column Value fields with the operator buttons to create the SQL clause.

   To create the first SQL statement to be added to the Where Clause field, complete the following steps:

   a. To define the software that the query will search for, enter SWARE_DESC in the Column Name field

      -OR-

      Click the ellipses (**...**) button to display the list of Available Columns, and select SWARE_DESC from the list of column names.

   b. Choose a logical operator to establish a relationship between the entry in Column Name and the entry in Column Value. For this query, leave the equals sign (=) as the logical operator, because the Column Value field will be used to specify the software for which the query should search. (If you want to search for machines that do not have a particular software, you would select Not.)

   c. Click the ellipses (**...**) button in the Column Value section. Select **Microsoft Word** and click the **Close** button, because the query should search for machines that have Microsoft Word associated with the machine ID. This completes the search criteria for this where clause.

      Note: If the selected table or view has a large number of rows, the Column Value dialog box might take a long time to appear.

d. Click the **Add** button to add the criteria to the Where Clause field.

10. To narrow the results further and return only machines that have at least 32 MB of memory, add the next SQL statement to the Where Clause field. Complete the following steps:

    a. In the Column Name field, enter the name of the column that stores information about physical memory, which is PHYSICAL_TOTAL_KB.

    b. From the menu next to the Column Value field, select **>=** as the logical operator so the query will return both machines with the specified amount of physical memory and machines with more than the specified amount.

    c. In the Column Value field, enter the minimum amount of memory that the machines should have, which is **32000 KB**. (Memory must be entered in KB.)

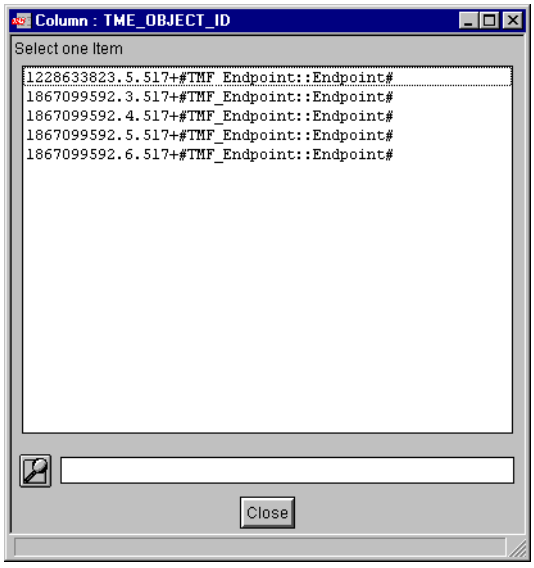
    d. Click the **Add** button to add the second where clause to the Where Clause field.

11. The final where clause for this query specifies that the query should return information only about machines in a particular TMR. To narrow the query to all the machines in a particular TMR, you specify that information should be returned only for machines with a TME_OBJECT_ID that includes the number of the TMR. To do this, complete the following steps:

    a. In the Column Name field, enter TME_OBJECT_ID.

    b. From the menu next to the Column Value field, select **LIKE** as the logical operator. This enables you to use a wildcard character (%) when defining the Tivoli object ID values for which the query should search.

    c. Click the ellipses button (**...**). The Column: TME_OBJECT_ID dialog box is displayed, which lists the TME_OBJECT_ID for each machine in the configuration repository. The first number of a machine's TME_OBJECT_ID is the region number of the TMR in which the object resides.



d. In the Column: TME_OBJECT_ID dialog box, select the TME_OBJECT_ID of a machine that includes the region number for the correct TMR and click the **Close** button. This populates the Column Value field with the Tivoli object ID.

   See "Using Queries in Interconnected TMRs" on page 8-20 for information about determining the numeric ID of the TMR. See "Querying for Information about One Client" on page 8-16 for instructions on viewing information about a particular client.

e. In the Column Value field, change the TME_OBJECT_ID to be the TMR number with a wildcard character, such as **105620229%**. To do this, delete the part of TME_OBJECT_ID after the TMR number and add a percent sign (%). The query will return results only for machines in the TMR that corresponds to that number. This specifies that any client in the specified TMR will be included in the query results.

f. Click the **Add** button. The final clause of the where clause is displayed in the Where Clause field.

12. In the Additional Clauses field, add the following clause:

```
INVENTORYDATA.COMPUTER_SYS_ID =
INST_SWARE_VIEW.COMPUTER_SYS_ID
```

Note: If you choose to query more than one table or view, you must create a join of the tables or views in the Additional Clauses panel.

13. Click the **Create & Close** button to create the query and return to the query library window.

    -OR-

    To create the query, click the **Create** button.

14. To run the query, click the **Run Query** button. For more information about running queries and saving results, see "Running a Query" on page 8-10.

Command Line

The following example uses the **wcrtquery** command to create the MSWord_Finder query from the previous section.

```
wcrtquery -d "Finds version of MSWord" \
-r inv_query \
-v INVENTORYDATA \
-v INST_SWARE_VIEW \
-c INST_SWARE_VIEW.SWARE_DESC -c INST_SWARE_VIEW.SWARE_VERS \
-c INST_SWARE_VIEW.TME_OJBECT_LABEL -c
INVENTORYDATA.PHYSICAL_TOTAL_KB \
-w "(INST_SWARE_VIEW.SWARE_DESC='Microsoft Word') and \
(INVENTORYDATA.PHYSICAL_TOTAL_KB>=32000) and \
(TME_OBJECT_ID like '1947203709%')" INVENTORY_QUERY MSWord_Finder
```

See the *Tivoli Management Framework Reference Manual* for more information about the **wcrtquery** command.

Running a Query

After you create a query to retrieve information about your Tivoli environment, you can run it and either view the results or save them in a file. To run any query and display or save the results, do one of the following:

- Right-click the icon of the query and select **Run Query**.
- Click the **Run Query** button on the Create Query dialog box
- Click the **Run Query** button on the Edit Query dialog box
- Use the **wrunquery** command

The following table provides the authorization roles required to perform this task:

Operation	Context	Role
Run a query	The policy region that contains the query library of the applicable query	**senior**, **super**, **Query_execute**, or **Query_edit**

You can run a query from either the Tivoli desktop or the command line.

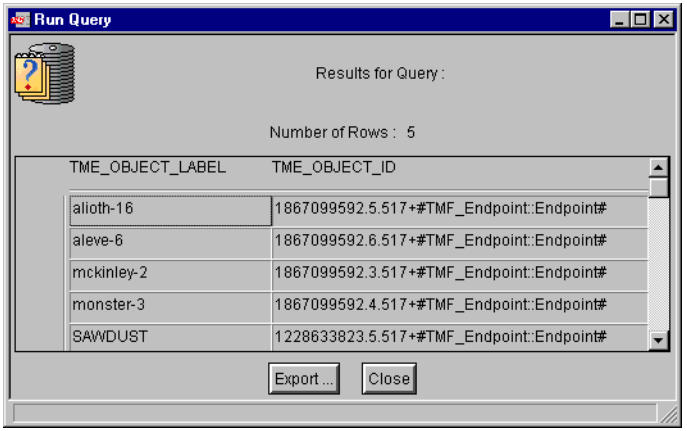
Desktop

In a query library, right-click the icon of a query and select **Run Query**.

-OR-

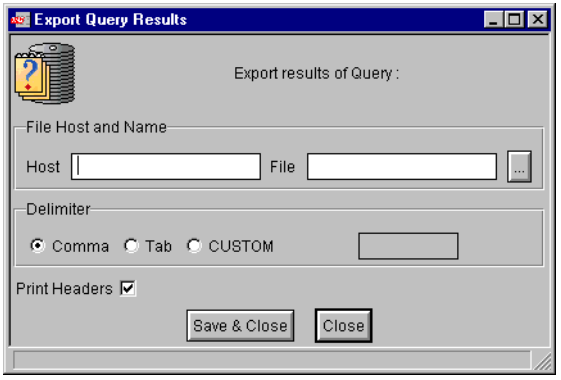
Select **Run Query** from either the Create Query or Edit Query dialog box.

The Run Query dialog box is displayed, showing the results of the query in tabular form.



If you want to save the results of the query, you can do so from the Run Query dialog box. Complete the following steps:

1. On the Run Query dialog box, click the **Export** button. The Export Query Results dialog box is displayed.



2. In the Host field, enter the name of the managed node on which you would like to save the results file. If you do not specify a managed node, you will receive an error message.
3. In the File field, enter a directory and name for the query results file.

-OR-

Click the ellipses **(…)** button to browse the file system and select a directory and file name.

4. Select one of the **Delimiter** options to specify how to separate entries in the query results file. If you want to use a delimiter other than a comma or a tab, select **Custom** and enter a delimiter in the field.
5. If you want the output file to include the name of the query and the names of the columns, select the **Print Headers** check box.
6. Click the **Save & Close** button to create the file.

Command Line

You can use the **wrunquery** command to run any query and either display the results to standard output or save the results in a file. To use the **wrunquery** command to run the Win-machines query and save the results in a file, enter the following:

```
wrunquery -n -h amon -f /tmp/query.txt -d ";" Win-machines
```

where:

-n

Omits headers from the results file.

-h amon

Specifies **amon** as the name of the managed node on which to store the file.

-f /tmp/query.txt

Specifies **/tmp/query.txt** as the location and name of the query results file.

-d ";"

Specifies a semi-colon as the delimiter.

Win-machines

Specifies the name of the query to be run.

See the *Tivoli Management Framework Reference Manual* for more information about the **wrunquery** command.

Defining Subscribers with Queries

In addition to using queries to find out about your Tivoli environment, you can also use queries to return a list of clients in your environment that meet certain criteria. For example, you can run a query to select targets for an inventory profile distribution. When you run the query, it selects the list of the clients in the policy region that meet the query criteria.

Tivoli Inventory provides you with several pre-defined queries that you can run to identify subscribers for an inventory profile. These queries belong to the SUBSCRIPTION_QUERY query library. For information about installing subscription queries, see “Installing Tivoli Inventory and Subscription Queries” on page 3-22. For a list and description of available subscription queries, see “Subscription Queries” on page C-80.

The following table provides the authorization roles required to perform these tasks:

Operation	Context	Role
Select targets for an inventory profile distribution	inventory profile	**senior** or **super**

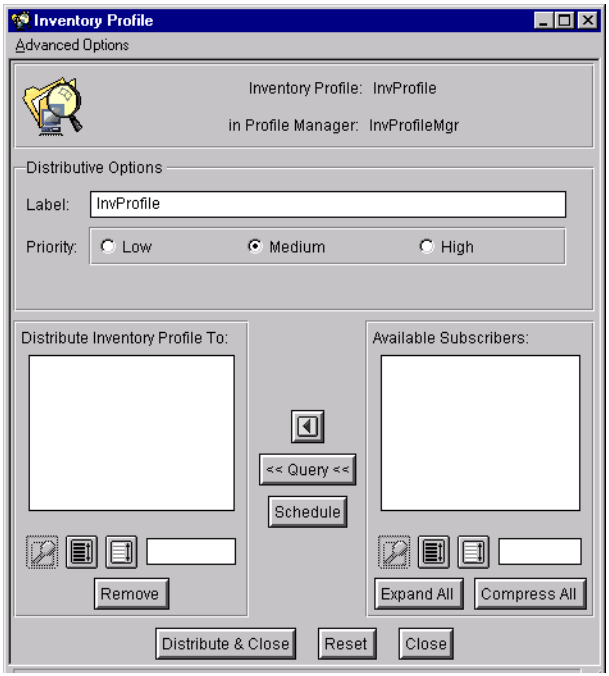
You can select targets from either the Tivoli desktop or the command line.

Desktop

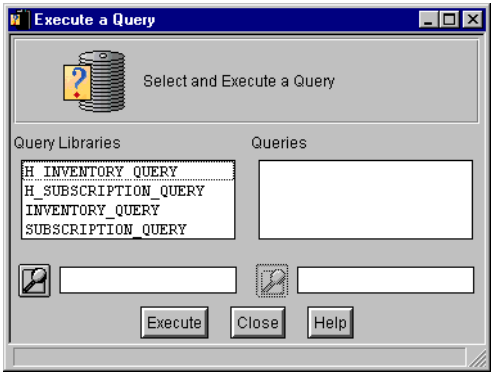
If you run a query from the Inventory Profile dialog box, the query must include the TME_OBJECT_ID and TME_OBJECT_LABEL columns in the Chosen Columns list.

Complete the following steps to define subscribers for a profile by running a query from the Inventory Profile dialog box:

1. From the pop-up menu of an inventory profile, select **Distribute**. The Inventory Profile dialog box is displayed.



2. Click the **Query** button. The Execute a Query dialog box is displayed.



3. In the Query Libraries scrolling list, double-click the query library that contains the query you want to run. The Query Libraries scrolling list shows a list of all the query libraries in the policy

region. The query library you select populates the Queries scrolling list.

4. In the Queries scrolling list, select a query.
5. Click the **Execute** button. The Execute a Query dialog box closes and the Inventory Profile dialog box is displayed. The query moves the subscribers in the policy region that meet the query criteria to the Distribute Inventory Profile To scrolling list.

Command Line

The **wruninvquery** command runs a query and returns the results in an IDL format that you can use with scripts to populate subscription lists. A query that you run with this command must include both the TME_OBJECT_ID and TME_OBJECT_LABEL columns. In contrast, the **wrunquery** command can be used with any query, and the results are sent to standard output or to a file.

To run the Win-machines query, enter the following:

```
wruninvquery -l Win-machines
```

where:

-l

Lists the names of the machines that match the query criteria instead of their object IDs.

Win-machines

Specifies Win-machines as the query to run.

See the *Tivoli Management Framework Reference Manual* for more information about the **wruninvquery** and **wrunquery** command.

Querying for Information about One Client

In addition to using queries to gather information about the Tivoli environment or to generate lists of targets, you can use queries to access inventory information about one client in the Tivoli environment. A query will return the inventory scan results for the specified machine. Any query you use to return information about one client must include the columns TME_OBJECT_ID and TME_OBJECT_LABEL.

The following table provides the authorization roles required to run a query from the Profile Manager dialog box:

Operation	Context	Role
Run a query about one client	The policy region that contains the query library of the applicable query	**RIM_view**, **Query_view**, **Query_execute**, **Query_edit**, **senior**, or **super**

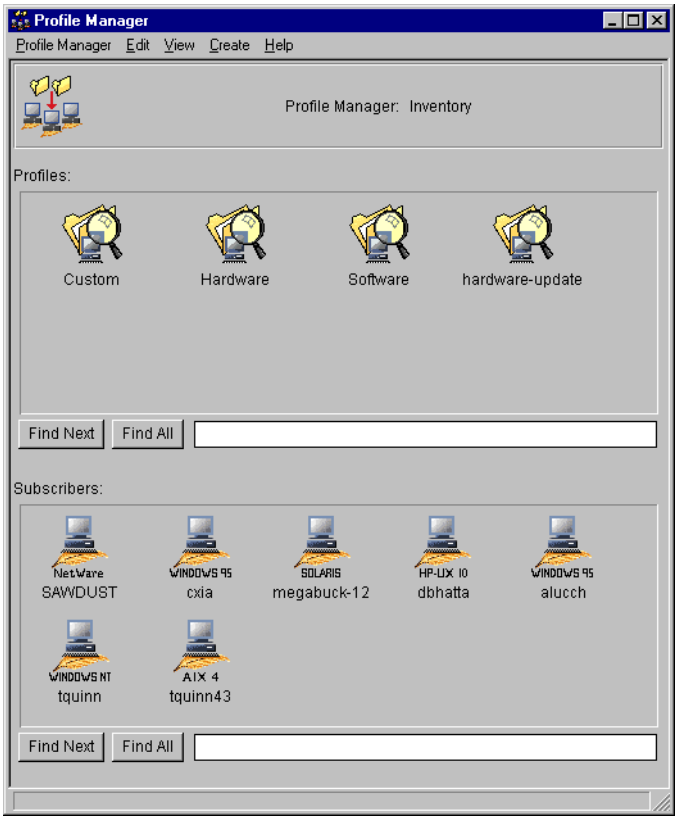
You can retrieve inventory information for a client from either the Tivoli desktop or the command line.

Desktop

Complete the following steps to display information from the icon menu of an endpoint:

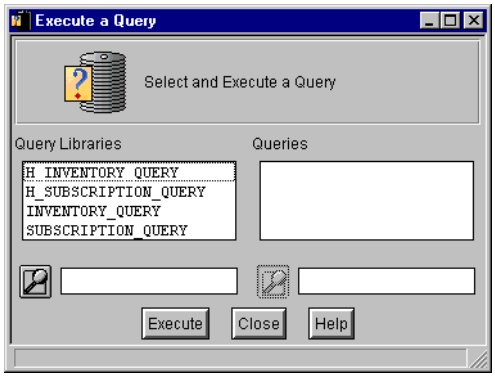
1. From the Profile Manager dialog box, right-click the icon of the desired endpoint.

   Note: Endpoint icons are visible only from within the window of the profile manager to which they are subscribed.



2. To see information about the endpoint, complete the following steps:

   a. Select **Execute Query** from the pop-up menu. The Execute a Query dialog box is displayed.



b. In the Query Libraries scrolling list, double-click the query library that contains the query you want to run. The Query Libraries scrolling list shows a list of all the query libraries in the policy region. The query library you select populates the Queries scrolling list.

c. In the Queries scrolling list, select a query.

d. Click the **Execute** button. Tivoli Inventory runs the query and displays results for only the selected machine.

Command Line

Use the **wqueryinv** command to return inventory information about a particular client.

```
wqueryinv -n -d ";" -h amon -f /tmp/x1.txt \
-q PARTITION_QUERY -m tacoma
```

where:

–n

Specifies that headers should not be included in the query results file.

–d ";"

Specifies that a semi-colon will be used to separate entries in the query results file.

–h amon

Specifies the name of the host where the query results file will be saved.

–f /tmp/x1.txt

Specifies the directory and name of the query results file.

–q PARTITION_QUERY

Specifies the name of the query to be executed.

–m tacoma

Specifies the Tivoli object label of the machine that you want to retrieve inventory information about and that it is a managed node.

See Appendix B, "Commands," for more information about the **wqueryinv** command.

Using Queries in Interconnected TMRs

This section explains how to query interconnected TMRs and display the data for each TMR separately. For example, some Tivoli Inventory users collect data for multiple internal or external customers. If the systems for each customer are in separate TMRs, you can use the following procedure to view each customer's data separately.

To view the data from each TMR separately, you create a query that separates the data using the object ID (OID). An OID is a three-part identifier of the form *n.n.n*, for example 1264987995.1.326. The OID is made up of the following values:

- The region number. Objects in Tivoli Management Framework, Version 3.0, and above have ten-digit region numbers (1264987995 in the example above). Region numbers are generated when the TMR server is installed. An algorithm is used to generate a region number for each TMR. In most cases, the number generated is unique. However, due to the random element in the region number generation, there is a small possibility that two TMRs could generate the same region number. If this happens,

one of the regions must be reinstalled to connect to the region with the same number.

- The object dispatcher (oserv) number. Sometimes referred to as the host number, dispatcher numbers are assigned in the order managed nodes are installed. The TMR server will be dispatcher 1, the next managed node installed will be 2, and so on.
- The object number. This is a number for the object itself.

When a managed node or endpoint is created in a given TMR, the region number of the managed node or endpoint matches that of the TMR server.

To find which systems are in a specific TMR, edit the INVENTORY_HWARE query to select only the columns TME_OBJECT_LABEL, TME_OBJECT_ID, and COMPUTER_SYS_ID. In the Additional Clauses text box, enter the following clause:

```
order by TME_OBJECT_ID
```

When you run this query, it returns each grouping of endpoints by TMR.

To query specific data for a TMR, edit the desired query by adding the following where clause:

```
TME_OBJECT_ID like 'nnnnnnnnn%'
```

where *nnnnnnnnn* is the first 10 digits of the OID. To query for data in multiple TMRs, add a where clause for each OID.

All installed hardware and software information for a given system is tied to a computer system ID. If you want to view information about a specific endpoint, add the following where clause:

```
COMPUTER_SYS_ID = 'computer_sys_id'
```

where *computer_sys_id* is the COMPUTER_SYS_ID value of the desired endpoint.

A

Authorization Roles

This appendix contains tables that describe the activities and roles you need to perform activities using Tivoli Inventory.

Inventory Roles

To perform activities using Tivoli Inventory, you must be a Tivoli administrator. Depending on the activities you perform, you must have one or more of the following roles in either the Tivoli Management Region (TMR) or in the policy region in which the operation is performed:

- **Inventory_end_user**
- **Inventory_view**
- **Inventory_query**
- **Inventory_scan**
- **Query_view**
- **Query_edit**
- **Query_execute**
- **RIM_view**
- **RIM_update**
- **user**
- **admin**

- **senior**
- **super**
- **Install_product**
- **Install_client**
- **restore**
- **backup**

Inventory Operations

The following table lists the authorization roles required to perform Tivoli Inventory-specific tasks:

Operation	Required Role
Install Tivoli Inventory	**Install_product** or **super**
Use UserLink for Tivoli Inventory Web functions	**Inventory_end_user**
Create an inventory profile	**senior** or **super**
Customize or edit an inventory profile	**admin**, **senior**, or **super**
View an inventory profile	**Inventory_view**, **user**, **admin**, **senior**, or **super**
Clone an inventory profile	**senior** or **super**
Delete an inventory profile	**senior** or **super**
Set inventory profile subscribers	**admin**, **senior**, or **super**
Distribute an inventory profile	**Inventory_scan** or **admin**
View inventory data in the configuration repository	**Query_view**, **RIM_view**, **user**, **admin**, **senior**, or **super**

Operation	Required Role
Use Tivoli Inventory administrator Web pages	**RIM_view**
Add, delete, or edit software signatures	**RIM_update**, **senior**, or **super**

Query Operations

The following table lists the authorization roles required to perform query operations:

Operation	Required Role
Create a query library	**senior** or **super**
Create a query	**admin**, **senior**, or **super**
Edit a query	**Query_edit**, **admin**, **senior**, or **super**
View a query	**Query_execute, Query_edit, Query_view**, **user**, **admin**, **senior**, or **super**
Execute a query	**Query_edit, Query_execute**, **admin**, **senior**, or **super**

B

Commands

This appendix lists and documents, in alphabetical order, the Tivoli commands related to Tivoli Inventory. The commands for the Tivoli query facility and the relational database management system (RDBMS) interface module (RIM) are documented in the *Tivoli Management Framework Reference Manual*.

Most Tivoli commands begin with a **w**, and often vowels are omitted to shorten the name of a command. Commands are usually developed using the **w+verb+object** syntax, which matches the way you might think of the action. For example, to set the global properties of an inventory profile, use the **wsetinvglobal** command. To view the global properties, use the **wgetinvglobal** command.

Using Tivoli Commands

It is often necessary or convenient to perform a Tivoli operation from the command line interface rather than from the desktop. For example, you might prefer to use the command line interface (CLI) for any of the following reasons:

- You do not have access to a Tivoli desktop.
- You want to group several Tivoli commands in a shell script or batch file.
- You prefer to run a command from within a script that performs multiple operations.

Command Line Syntax

The reference pages in this appendix use the following special characters to define the command syntax:

[ ]	Identifies optional options. Options not enclosed in brackets are required.
...	Indicates that you can specify multiple values for the previous option.
|	Indicates mutually exclusive information. You can use either the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use of the command.
{ }	Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([ ]).

Example

wruninvquery [**–i**] [**–T** *idl-type*] [**–l | –t**] *query_name* [*input*]…

The ellipses (…) following the *input* option indicate that you can specify multiple query inputs. Also, if you choose to specify **–l** or **–t**, you can specify only one. These options are delimited by the **|** (logical or), indicating that they are mutually exclusive. You must specify the *query_name* option; all other options are optional, as denoted by the brackets ([ ]).

Object References

When an object is referenced from a command, the reference is not the absolute object reference that is used in programming. Instead, a user-friendly name is used. This user-friendly name derives from a name given to the object by the user of the application, such as when a policy region is created.

There are two different forms of names that can be used with CLI commands:

- Registered names
- Object paths

Tivoli commands support both naming schemes. Sometimes you will find it more convenient to use one form over the other. If you receive an error message indicating that a resource cannot be found, try a different naming convention.

Registered Names

The Tivoli name registry contains registered names. A *registered name* is the name by which a resource instance is registered with the name registry when it is created. Every resource has a name and is of some particular type. For example, a printer called lp01 has a name lp01 and is of type printer. Some examples of registered names used as options for the **wls** and **wmv** commands are as follows:

```
wls @PolicyRegion:Servers
wmv @ManagedNode:ayers-rock @PolicyRegion:Servers
```

The syntax for specifying a resource using the registered name facility is *@type:name*, where *type* is the resource type and *name* is the particular instance of that resource on which you want to perform some operation.

The name registry does not allow two resources of the same type to have the same name within a single Tivoli Management Region (TMR). However, it is possible for resource names to be duplicated within two or more connected TMRs. If you attempt to perform an action on a resource with a duplicated name, an error message is returned, and the action is not performed. To avoid this situation, you should either rename one of the resources or differentiate between the resources by appending a region name to the resource name, as follows:

```
wls @ManagedNode:moria#moria-Region
```

For more information about the **wls** and **wmv** commands, see the *Tivoli Management Framework Reference Manual*.

Object Paths

Object paths are similar to path names in file systems and can be relative or absolute. An absolute path is one that starts with a slash (/) character. A relative path can start with any character, including the special path components '.' and '..'. Some examples of object path names used as options for the **wls** and **wmv** commands are as follows:

```
wls /Regions/Servers
wmv../Servers/ayers-rock /Regions/Servers
```

The syntax for specifying a resource using an object path as a name is */distinguished/parent/*[*type*:]*name*, where *distinguished* is a resource type, *parent* is the start of the object path name, *type* is used to further identify a resource, and *name* is the particular instance on which you want to perform some operation. You often use the optional type qualifier when you need to name a particular resource that has the same name as some other resource of a different type.

Examples

If an example of a command is too long to fit within the text margins of this document, a backslash (\) is used to break the line. This indicates that you must not break the line when you actually enter the command.

Tivoli Inventory Commands

Tivoli Inventory provides a command line interface for many of the same operations that you can use from the desktop. Tivoli Inventory includes the following commands:

Command	Purpose
wdistinv	Distributes an inventory profile.
wepscan	Initiates a scan from an endpoint.
wgetinvglobal	Lists the global properties that are set for an inventory profile.

Command	Purpose
wgetinvpcfiles	Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.
wgetinvpchw	Lists the options that are set for a hardware scan of a PC endpoint.
wgetinvpcsw	Lists the options for PC software scans that are configured for an inventory profile.
wgetinvunixfiles	Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints.
wgetinvunixhw	Lists the options that are set for a hardware scan of a UNIX endpoint.
wgetinvunixsw	Lists the options for UNIX software scans that are configured for an inventory profile.
wcrtinvcb	Creates the inventory callback object.
winvfilter	Modifies files in the custom filter list stored in the configuration repository.
winvrmnode	Removes from the configuration repository all scanned information about a system.
winvsig	Modifies the list of software signatures in the configuration repository.
wqueryinv	Queries the configuration repository for information about an endpoint.
wsetinvglobal	Sets the global properties for an inventory profile.
wsetinvpcfiles	Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.
wsetinvpchw	Sets the options for hardware scans of PC endpoints.

Command	Purpose
wsetinvpcsw	Sets the options for software scans of PC endpoints.
wsetinvunixfiles	Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints.
wsetinvunixhw	Sets the options for hardware scans of UNIX endpoints.
wsetinvunixsw	Sets the options for software scans of UNIX endpoints.

SCS-Related Commands

The following table describes the commands related to Scalable Collection Service (SCS) that you can use with Tivoli Inventory. Currently, you can perform SCS tasks only from the CLI.

Command	Purpose
wcollect	Returns information about the current configuration of a collector, starts and stops a collector, resets collection routes that are cached at the collector, and configures options for a collector.
wcrtinvcb	Creates an instance of the inventory callback object. (You can create only one instance per TMR.)
wcrtinvdh	Creates an instance of the inventory data handler object. (You can create only one instance per TMR.)
wcstat	Returns status information about a collector, returns information about a collection table of contents (CTOC), and returns the contents of a collector's queues.
wgetinvdh	Returns configuration information about the inventory data handler.

Command	Purpose
wgetscanstat	Returns information about current inventory scans.
wsetinvdh	Changes the settings of the inventory data handler.

These commands fall into two major groups:

- Commands for SCS components

  You use these commands to configure and get information about collectors. These commands include **wcollect** and **wcstat**. (You can also use the **wcollect** command to configure some options for the inventory data handler.)

- Commands for Tivoli Inventory components

  You use these commands to control how Tivoli Inventory works with SCS, and to gather SCS-related information from Tivoli Inventory components. These commands separate into two categories:

  - Commands you use with inventory profiles. These commands include **wgetinvglobal** and **wsetinvglobal**.
  - Commands you use with the inventory data handler and status collector. These commands include **wcrtinvdh**, **wgetinvdh**, **wgetscanstat**, and **wsetinvdh**.

Command Syntax

This section lists Tivoli Inventory commands, with syntax and descriptions of their functions. You can also access these listings by using the **man** command on UNIX systems.

wcollect

Returns information about the configuration of a collector, starts and stops a collector, resets collection routes that are cached in a collector, and configures options for a collector.

Syntax

wcollect [*options*] *collector*

where *collector* is of the form **@ManagedNode:***collector_name*, **@Gateway:***collector_name*, or **@InvDataHandler:inv_data_handler** (*collector_name* is the name of the managed node or gateway)

The *options* are from one of the following categories:

- Options for active collectors

  [**–s** | **–h immediate** | **graceful**]

- Options for collector configuration and tuning

  [**–d 0** | **1** | **2** | **3**]

  [**–g** *debug_log_size*]

  [**–l** *runtime_location*]

  [**–z** *depot_size*]

  [**–c** *depot_chunk_size*]

  [**–i** *thread_idle_down_time*]

  [**–p** *thread_sleep_time*]

  [**–t** *max_input_threads*]

  [**–m** *max_input_retries*]

  [**–o** *max_output_threads*]

  [**–e** *retry_delay_time*]

  [**–r**]

  [**–x** *offlinks_range_to_prohibit_collection*]

  [**–f true** | **false**]

Description

The **wcollect** command has three main functions. First, you can obtain information about the configuration of a collector by running the command and specifying the collector for which you want information.

Second, you can use the **wcollect** command to stop or start an existing collector or to delete cached routing information stored in the collector. Options that allow you do this are referred to as active-collector options.

Third, you can use the **wcollect** command to configure collector attributes. Options that allow you to do this are referred to as collector-configuration options.

Options

The options can be divided into options for active collectors and options for collector configuration.

Active Collectors

–s Starts the collector after it has been halted.

–h *option* Halts the collector. Use one of the following options:

- **graceful**—The collector stops after completing any remaining active collections.
- **immediate**—The collector stops without waiting for active collections to finish processing.

Collector Configuration

–d *option* Specifies the level of debugging information to log in the SCS log file. Use one of the following options:

- **0**—Turns off logging.
- **1**—Logs only fatal errors.
- **2**—Logs fatal errors and warning messages.
- **3**—Logs all debugging messages.

The default value is 1. The SCS log file mcollect.log resides in the $DBDIR/mcollect directory of each collector.

–g *debug_log_size*

Specifies the maximum size in megabytes (MB) of the collector log file mcollect.log. When the file reaches the maximum size, it discards 90 percent of its contents and keeps the most recent 10 percent. By default, the maximum size is 1 MB.

–l *runtime_location*

Specifies the location of the run-time directory for the collector. The run-time directory contains the depot and run-time (*.dat and *.log) files. This directory must reside on a stable disk with an amount of free space large enough to ensure persistent storage of collections. The run-time directory should not be a temporary directory. You must also provide read-write privileges for the **tmersrvd** or **nobody** account (the Tivoli unprivileged account) in this directory. By default, the run-time directory is at $DBDIR/mcollect on each collector, and the depot is at $DBDIR/mcollect/depot.

Note: After you specify a new run-time directory, you must stop the collector, and then move the depot and any data from the old run-time directory to the new directory.

–z *depot_size* Specifies the size of the depot directory in MB. By default, this value is set to 40 MB. You can make the depot larger than its previous size, but you cannot make it smaller. For example, you can set the depot size to 50 MB. Later, you can set the size to 55 MB, but you cannot set it to 45 MB.

–c *depot_chunk_size*

Specifies the size of the transmission chunk in kilobytes (KB). When a downstream collector sends data to the next collector, it sends the data in separate units called transmission chunks. The default chunk size is 1024 KB.

This value should be large enough to keep the transfer of data efficient, but small enough to make most efficient use of the available bandwidth.

–i *thread_idle_down_time*

Specifies the number of seconds that a thread can be idle before it is shut down. The default idle time is 60 seconds.

–p *thread_sleep_time*

Specifies the number of seconds that a thread *sleeps* (waits) if system or network limitations have been reached. The thread attempts to resume the collection process after this period is completed. The default value is 5 seconds.

–t *max_input_threads*

Specifies the maximum number of input threads that the collector can process concurrently. This number should be large enough to maximize collector efficiency, but small enough to avoid excessive consumption of system resources. The default value is 5.

–m *max_input_retries*

Specifies the maximum number of attempts to process a collection request from a downstream collector. The default value is 10. You can set the time to wait before trying again to process a request by using **–e** *retry_delay_time*.

–o *max_output_threads*

Specifies the maximum number of output threads that the collector can process concurrently. The default value is 5. This number should be large enough to maximize collector efficiency, but small enough to avoid excessive consumption of system resources. It is recommended that this value match the total number of RDBMS connections set for all RIM objects used by the inventory data handler. For more information about setting RDBMS connections for RIM objects, see "Configuring RIM Objects" on page 4-13.

–e *retry_delay_time*

Sets the time in seconds to wait before trying again to process an input or output request to the collector. The

default value is 1 second. You can set the maximum number of attempts by using **–m** *max_input_retries*.

–r Removes locally cached collection routes on each collector. After you execute this command, when the collector processes the next collection request, the collector loads new routing information from the collection manager.

Run this option on all affected collectors after you make any modifications to the collection hierarchy using the **wrpt** command. Doing this synchronizes the routing information stored on those collectors.

–x *offlinks_range_to_prohibit_collection*

Specifies the range of offlinks. Offlinks are the object dispatcher numbers of collectors from which the specified collector must not collect data. In other words, you use this option to turn off the links to the collectors whose object dispatcher numbers you enter.

You can list the object dispatcher numbers in ascending numeric order, separated by commas, as shown in the following example:

```
wcollect -x "4,5,6,7" collector
```

Or, you can use a dash to indicate a range of object dispatcher numbers:

```
wcollect -x "4-7" collector
```

By setting this option, you can control network traffic between downstream collectors. For example, you can turn off links to some collectors during busy hours, and then restore the links when network traffic is light. This option does not affect data collection between endpoints and the first gateway collector.

To turn on all the links that you have previously turned off with this option, specify a null string (" ") as the value.

By default, no object dispatcher numbers are specified.

–f *option* Specifies whether information about completed CTOCs should be written to the log file. In any stage of the collection, CTOCs can be tracked using the **wcstat** command. Use one of the following options:

true	Information about completed CTOCs is written to the log file. This is the default option.
false	Information about completed CTOCs is not written to the log file.

@Gateway:*collector_name*
Specifies the name of the collector on which to run this command. Use **@Gateway:***collector_name* for collectors that are on gateways.

@ManagedNode:*collector_name*
Specifies the name of the collector on which to run this command. Use **@ManagedNode:***collector_name* for collectors that are on managed nodes that are not configured to be gateways.

@InvDataHandler:inv_data_handler
Specifies that this command is to be run on the inventory data handler.

Authorization

senior

Notes

You can execute collector-configuration options at any time, but you must restart the affected collectors for the changes to take effect. You halt and restart collectors using the **–h** and **–s** options of the **wcollect** command.

Examples

The following examples illustrate various operations you can complete with the **wcollect** command.

Request Information about Current Configuration

The following example returns information about the current configuration for a collector:

```
wcollect @ManagedNode:aztlan
```

The output is as follows:

```
Collector: @ManagedNode:aztlan
        debug_level = DEBUG (all messages)
        debug_log_size = 2 MB
        runtime_location = /data/aztlan/aztlan.db/mcollect
        depot_size = 40 MB
        depot_chunk = 1024 KB
        thread_idle_down_time = 60 seconds
        thread_sleep_time = 5 seconds
        max_input_threads = 5
        max_input_retries = 10
        max_output_threads = 5
        retry_delay_time = 1 seconds
        offlinks =
        log_completed_ctoc = true
```

Stop a Collector After Collections Complete

The following command stops a collector after processing is complete on all active collections:

```
wcollect –h graceful @Gateway:drodriguez-gateway
```

Configure a Collector

The following command specifies the amount of debugging information to log and the location and size of the depot for a collector.

```
wcollect –d 3 –l /tmp/dionicio/depot –z 80 @ManagedNode:aztlan
```

Note: After running this command, you must stop the collector, and then move any data from the old depot directory to the new directory.

Turn Off Links to a Collector

The following example turns off the links from the collector on aztlan to object dispatchers 2, 5, 6, 7, 8, and 11. Therefore, the collector on aztlan cannot collect data from those systems. (This command does not affect data collection between endpoints and the first gateway collector.)

```
wcollect -x "2,5-8,11" @ManagedNode:aztlan
```

Turn On All Links to a Collector

The following example turns on the links to all systems for which links were previously turned off.

```
wcollect -x "" @ManagedNode:aztlan
```

See Also

wcstat, **wrpt** (in the *Tivoli Management Framework Reference Manual*)

wcrtinvcb

Creates the inventory callback object.

Syntax

wcrtinvcb *managed_node*

Description

This command creates the inventory callback object. When you install Tivoli Inventory, the inventory callback object is created automatically on the managed node that you specify. Use this command to create an instance of the inventory callback object, for example to move the inventory callback object to a different managed node. For the procedure to do this, see "Moving the Inventory Callback Object" on page 4-10. You can create the inventory callback object on any managed node in your TMR. The inventory callback object performs the following functions:

- When inventory data cannot be collected using the collector hierarchy, MDist 2 sends the data to the inventory callback object, which sends it to the inventory data handler. The data is then sent through one or more RIM objects to the configuration repository.
- For all scans, MDist 2 sends a status message to the inventory callback object indicating whether it successfully delivered the inventory profile to the endpoint. This message indicates only that the endpoint processed the profile, not that the scan data successfully reached the configuration repository.

Options

managed_node

The name of the managed node on which you want to create the inventory callback object.

Authorization

admin

Examples

The following example creates the inventory callback object on managed node fuji.

```
wcrtinvcb fuji
```

wcrtinvdh

Creates an instance of the inventory data handler and sets some options for the inventory data handler.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wcrtinvdh –d *status_directory managed_node*

wcrtinvdh –n *bundle_every_n_minutes managed_node*

wcrtinvdh –q *bundle_every_n_targets managed_node*

wcrtinvdh –r *max_RDBMS_retries managed_node*

wcrtinvdh –s {**YES** | **NO**} *managed_node*

wcrtinvdh –t *RDBMS_retry_delay_time managed_node*

Description

This command creates an instance of the inventory data handler, specifies the interval at which scan completion notifications are sent, and sets the number of retries and the timeout period for writing inventory information to the configuration repository.

You use the **wcrtinvdh** command to create the inventory data handler instance @InvDataHandler:inv_data_handler in the Tivoli object database on the managed node that you specify. The inventory data handler is the Tivoli Inventory object that receives data from collectors and sends the data to the configuration repository. The inventory data handler is created automatically during installation of Tivoli Inventory. Use this command to create an instance of the inventory data handler, for example to move the inventory data handler to a different managed node. For the procedure to do this, see “Moving the Inventory Data Handler” on page 4-10. You can have only one instance of the inventory data handler object in a TMR.

In addition, you can use the **wcrtinvdh** command to specify the interval at which scan completion notifications are sent. You can specify this interval only for inventory profiles in which the **–n** option for **wsetinvglobal** is set to **BUNDLE**.)

You also use the **wcrtinvdh** command to set the number of retries and the timeout period for writing to the configuration repository.

Options

–d *status_directory*

Specifies where the inventory data handler stores status information that can be restored in case of a system failure. By default, the location is **$DBDIR/inventory/stat_dir** on the managed node where the inventory data handler resides.

–n *bundle_every_n_minutes*

Specifies the interval at which scan completion notifications are sent (when the **–n** option for the **wsetinvglobal** command is set to **BUNDLE**). If you set this number, Tivoli Inventory sends a notice with a list of the targets on which scans have completed during the specified time period. The default value is 10 minutes.

If no scans complete in the specified time period, no notification is sent.

If you set this option to 0, notification occurs according to the value that you set for the **–q** option of the **wcrtinvdh** or **wsetinvdh** command.

–q *bundle_every_n_targets*

Specifies the maximum number of targets in a bundle. A bundle is a group of targets about which status information is sent at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile.

The default value is 10 targets.

This option, in combination with the **–n** option of the **wsetinvglobal** command, configures status information to be sent in bundles.

Notes

- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.
- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.

–r *max_RDBMS_retries*
Specifies the number of times the inventory data handler tries to write data to a RIM object. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.

–s {**YES** | **NO**} Specifies whether the inventory data handler stores status information that can be restored in case of a system failure. If you set this option to **YES**, status information is stored. If you set this option to **NO**, status information is not stored. By default, status information is stored.

–t *RDBMS_retry_delay_time*
Specifies a value from which to calculate the timeout period in seconds between retries of writes to a RIM object. This timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

The default value is 30 seconds.

managed_node Specifies the managed node that will act as the inventory data handler.

Authorization

admin, **senior**, or **super**

Examples

The following example creates the inventory data handler on the managed node mckinley, sets the time interval for notification to 20 minutes (for inventory profiles with notification set to **BUNDLE**—see the **wsetinvglobal** command), and sets the number of retries for RIM failures to 3.

```
wcrtinvdh –n 20 –r 3 mckinley
```

See Also

wcollect, **wgetinvdh**, **wgetinvglobal**, **wsetinvdh**, **wsetinvglobal**

wcstat

Returns status information about a collector, returns information about a CTOC, and returns the contents of a collector's queues.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wcstat *collector*

wcstat –v *ctoc_id collector*

wcstat –q [**ioecd**] *collector*

where *collector* is of the form **@ManagedNode:***collector_name*, **@Gateway:***collector_name*, or **@InvDataHandler:inv_data_handler**

Description

With the **wcstat** command, you can complete the following tasks:

- Retrieve status information about a specified collector.
- Retrieve information about a specified CTOC on a collector.
- Retrieve the contents of any or all of a collector's queues.

Options

collector — Specifies the name of the collector for which you want to run the **wcstat** command. You must use one of the following formats:

- **@Gateway:***collector_name* where *collector_name* is the name of a gateway that is a collector
- **@ManagedNode:***collector_name* where *collector_name* is the name of a managed node that is a collector

–v *ctoc_id* — Specifies the CTOC for which you want status information returned.

–q *option* Specifies the type of queue to be returned. The following options return the contents of the specified queue. You can enter any combination of these options.

- **i**—input queue
- **o**—output queue
- **e**—error queue
- **c**—completed queue
- **d**—deferred queue

Authorization

senior

Examples

The following examples illustrate operations you can complete with the **wcstat** command.

Return the Contents of a Completed Queue

The following example returns the contents of the completed queue for the collector aztlan:

```
wcstat –q c @Gateway:aztlan–gateway
```

The output is as follows:

```
CTOC ID: wepm_ctoc_928780348
CTOC Properties:
    PRIORITY: 1
    COLL_STATUS: TRUE
    SOURCE_NAME: drodriguez2
    SOURCE_OID: 2112331601.2.19
    SOURCE_METHOD: mc_get_data
    DEST_OID: 2112331601.1.675
    INV_DDC::InvDataHandler
    DEST_METHOD: mc_request_collection
    DATAPACK: 2129
Client Properties:
    scan_id: 2147483647
Collection Status: CTOC_DONE
#Retries: 0
```

Return Status Information of a CTOC

The following example returns information about the specified CTOC:

```
wcstat –v ctoc3_11836_9110 @ManagedNode:calypso
```

The output is as follows:

```
CTOC ID: ctoc3_11836_9110
CTOC Properties:
    PRIORITY: 1
    COLL_STATUS: OK
    SOURCE_OID: 1637823410.2.19
    DEST_OID: 1637823410.1.552#MCFTP::Server#
    DEST_METHOD: callback_method
    DATAPACK: 33637364
Client Properties:
    MC_DEST_DIR: /tmp
Collection Status: QUEUED_OUTPUT
#Retries: 0
```

See Also

wcollect

wdistinv

Distributes an inventory profile.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wdistinv –l *mdist2_token=value* **@InventoryConfig:***profile_name*

wdistinv –T *subscribers_file* **@InventoryConfig:***profile_name*

wdistinv @InventoryConfig:*profile_name* **@Endpoint:***target1 ...*

Description

The **wdistinv** command enables you to distribute an inventory profile from the command line. If you do not specify targets for the distribution, the inventory profile is distributed to all subscribers of the profile manager in which the inventory profile resides.

Options

–l *mdist2_token=value*

Sets a value for the MDist 2 distribution options. You can specify multiple options by repeating **–l** *mdist2_token=value* multiple times. You can use the following tokens:

label=*description_string*

Specifies a description string for the distribution. The default value is the name of the inventory profile.

priority=h | m | l

Specifies the priority level, which is the order in which distributions are

handled by repeaters. You can set priority to one of the following values:

h	Highest priority.
m	Medium priority. This is the default value.
l	Low priority.

notify_interval=*minutes*

Specifies the interval, in minutes, that each repeater bundles and returns the completed results of the MDist 2 distribution. Use a positive integer. The default value is 1 minute.

send_timeout=*seconds*

Specifies the length of time a repeater waits for a target system to receive a block of data (for example, a repeater distributing an inventory profile to an endpoint). This timeout is used to detect network or endpoint failures. If a timeout occurs, the distribution remains in the repeater's queue and a retry occurs according to the **conn_retry_interval** value set with the Tivoli Management Framework **wmdist** command.

execute_timeout=*seconds*

Specifies the length of time a collector waits for an endpoint to return results (the downcall returns) after all of the data has been sent.

deadline=*mm/dd/yyyy hh:mm*

Specifies the date on which a distribution expires (the date that it fails for unavailable target systems). The default date is 72 hours from the time the profile is distributed.

mandatory_date=*mm/dd/yyyy hh:mm*

Specifies the date by which the distribution must be made. Distributions to roaming targets (such as laptops) can be deferred up to this date. When the date is reached, the profile is automatically distributed to all roaming targets that have not yet accepted it. The default value is 60 hours from the time the profile is distributed.

force_mandatory=y | n

Controls the way in which mandatory distributions on roaming endpoints are treated once the mandatory date is passed. Specify one of the following options:

- **y**—The mandatory distribution is automatically started as soon as the roaming user connects. This is the default option.
- **n**—The roaming user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n***=***mm/dd/yyyy hh:mm*

Specifies the date on which a reminder message must be sent to roaming targets that have not yet received the profile. For the value *n* you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order and you

cannot skip a number. For each escalation date, you must also specify an escalation message using **escalate_msg**.

escalate_msg_*n*

Specifies a message that must be sent to roaming targets that have not received the profile by the associated escalation date. For the value *n* you can specify a number, 0 through 9, which enables you to create up to ten messages. For *n*, you must specify the values in ascending numerical order and you cannot skip a number. For each escalation message, you must also specify an escalation date using **escalate_date**. Enter a message using the following format:

```
escalate_msg_n="message text"
```

hidden=y | n Indicates whether a distribution to a roaming endpoint is hidden. Hidden distributions are not revealed to the roaming user and are automatically distributed to the roaming endpoint. Distributions that are not hidden are visible to the roaming user, and the roaming user can defer the distribution. Use one of the following options:

- **y**—The distribution is hidden.
- **n**—The distribution is not hidden.

If you set this argument to y, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints=y | n

Indicates whether the operation defined in the **wdistinv** command supports mobile or roaming endpoints such as laptops. Use one of the following options:

- **y**—The distribution is to be transferred to any gateway where the roaming endpoint connects.
- **n**—After the profile is queued at a gateway it cannot be transferred to another gateway.

wake_on_lan=y | n

Indicates whether the operation sends a wake-on-LAN message to trigger the start of a system that is not available at the time an inventory profile is distributed. This option works only on systems that have a network card that is enabled for wake-on-LAN. Use one of the following options:

- **y**—Send a wake-on-LAN message.
- **n**—Do not send a wake-on-LAN message.

–T *subscribers_file*

Specifies a text file that contains a list of targets for the distribution. This file can list endpoints, profile managers, or both. Within this file, you must list the targets in one of the following formats:

- **@Endpoint:***endpoint_name*
- **@ProfileManager:***profile_manager_name*

For the *subscribers_file* value, you can list files using either relative or absolute path names. You can specify one or more file names. Separate multiple file names

with blank spaces. You can use this option alone or you can specify one or more targets for the distribution using the **@Endpoint:***target* option.

@InventoryConfig:*profile_name*
Specifies the profile on which you want to run the **wdistinv** command.

@Endpoint:*target*
Specifies the endpoint to which you want to distribute the inventory profile.

Authorization

Inventory_scan, **senior**, or **super**

Examples

The following example provides a label and sets a mandatory date for the distribution of profile InvProfile:

```
wdistinv -l label="Inventory Distribution" \
-l mandatory_date=04/06/2001 12:00 @InventoryConfig:InvProfile
```

The following example specifies a file containing a list of targets and sets the priority for the distribution of profile InvProfile:

```
wdistinv -T /tmp/endpoint.list -l priority=high \
@InventoryConfig:InvProfile
```

See Also

wdistrib, **wmdist**, **wrpt** (in the *Tivoli Management Framework Reference Manual*)

wepscan

Initiates a scan from an endpoint.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wepscan

wepscan –c

wepscan –d

wepscan –s

Description

The **wepscan** command enables you to initiate a scan from an endpoint, rather than having to distribute an inventory profile to the endpoint from a remote system. You can specify whether to scan the endpoint, send the scan data to the configuration repository, or both. The **wepscan** command can also generate a log file that contains the scan data or the contents of the configuration file.

Before you run **wepscan**, you must first set up your environment to access shared libraries needed by the command. To do this, you must run the lcf_env script. Depending on the platform type of the endpoint, this script is one of the following: lcf_env.bat, lcf_env.cmd, lcf_env.csh, or lcf_env.sh. This script is located either in the directory in which the endpoint is installed, or in the dat/1 subdirectory within that directory.

Solaris users must set the LD_LIBRARY_PATH environment variable before using **wepscan**. For example, a Bourne or Korn shell user would set the environment value as follows:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
export LD_LIBRARY_PATH
```

HP-UX users must set the SHLIB_PATH environment variable before using **wepscan**. For example, a Bourne or Korn shell user would set the environment value as follows:

```
SHLIB_PATH=$SHLIB_PATH:.
export SHLIB_PATH
```

You must run the **wepscan** command from the $LCF_BASE_DIR/inv/SCAN directory, where $LCF_BASE_DIR is the directory in which you installed the endpoint.

When you run the **wepscan** command with no options, it reads the configuration file, config.dmp, that was last distributed to the endpoint, and then runs the scan according to the options in the configuration file. It then creates a Management Information Format (MIF) file, parses the MIF file, and creates an encoded and compressed data file, INV_SA.DAT. The data file is stored on the endpoint.

Options

–c Reads the contents of the configuration file, config.dmp, and prints the contents to a log file named sa_config.log. This log file is saved in the directory from which you ran **wepscan**.

–d Runs a scan, saves the scan data on the endpoint, and creates a log file named sa_results.log that contains the scan data. This log file is saved in the directory from which you ran **wepscan**. In addition to the sa_results.log file, the **–d** option also creates the sa_config.log file that is created using the **–c** option.

–s Sends the scan data (the INV_SA.DAT file) to the configuration repository.

Authorization

This command is run locally at the endpoint. No Tivoli authorization is required.

Notes

This command does not work on endpoints of NetWare or OS/2 gateways.

Examples

The following example scans an endpoint using the scan options specified in the configuration file on the endpoint, and then stores the scan data on the endpoint:

```
wepscan
```

The following example sends scan data stored on the endpoint to the configuration repository:

```
wepscan -s
```

The following example prints the contents of the configuration file, config.dmp, to sa_config.log:

```
wepscan -c
```

wgetinvdh

Returns configuration information about the inventory data handler.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvdh –a

wgetinvdh –d

wgetinvdh –n

wgetinvdh –q

wgetinvdh –r

wgetinvdh –s

wgetinvdh –t

Description

The **wgetinvdh** command returns information about the inventory data handler, including how notification occurs, the number of retries, and the timeout period for writing data to a RIM object.

Options

–a	Returns all available information about the inventory data handler.
–d	Returns the value of *status_directory* as set by the **wcrtinvdh** or **wsetinvdh** commands. This directory stores status information so that it can be restored in case of a system failure.
–n	Returns the value of *bundle_every_n_minutes* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the interval in minutes at which scan completion notifications are sent.
–q	Returns the value of *bundle_every_n_targets* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the maximum number of targets in a bundle.

A bundle is a group of targets about which status information is sent to the Tivoli Inventory notice group at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile. This option as set by the **wcrtinvdh** or **wsetinvdh** commands, in combination with the **–n** option of the **wsetinvglobal** command, configures how status information is sent in bundles.

–r Returns the value of *max_RDBMS_retries* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies the number of times the inventory data handler tries to write data to a RIM object before returning a failure for the target to which the data is associated.

–s Returns information about whether the inventory data handler stores status information in case of a system failure. If YES is returned, status information is stored. If NO is returned, status information is not stored.

–t Returns the value of *RDBMS_retry_delay_time* as set by the **wcrtinvdh** or **wsetinvdh** commands, which specifies a value used to calculate the timeout period in seconds between retries of writes to a RIM object. This timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run this command with no options to return all configuration information about the inventory data handler.

Examples

The following example returns all available information about the inventory data handler:

```
wgetinvdh –a
```

The output is as follows:

```
Save status in the directory:
/Tivoli/db/fuji.db/inventory/stat_dir
Send bundled notification every: 10 minutes
Send bundled notification every: 10 targets
Max retries for RDBMS errors:      5
Save status in case of failure:  YES
Retry delay time for RDBMS errors: 30 seconds
Managed Node:                      fuji
```

See Also

wcollect, **wcrtinvdh**, **wsetinvdh**, **wsetinvglobal**

wgetinvglobal

Lists the global properties that are set for an inventory profile.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvglobal *profile_name*

wgetinvglobal –d *profile_name*

wgetinvglobal –e *profile_name*

wgetinvglobal –f *profile_name*

wgetinvglobal–h *profile_name*

wgetinvglobal –l *profile_name*

wgetinvglobal –m *profile_name*

wgetinvglobal –n *profile_name*

wgetinvglobal –t *profile_name*

wgetinvglobal –u *profile_name*

wgetinvglobal –w *profile_name*

Description

The **wgetinvglobal** command enables you to view the global properties that are set for an inventory profile. The information that this command returns depends on the options that you specify when you run it.

Options

–d Returns information about the distribution options for the specified inventory profile. The following values can be returned:

- **CONFIG** Distribute the configuration file only.
- **ALL** Distribute the configuration file and run the scan.

–e Returns the time, in seconds, that a profile attempts to scan an endpoint before it times out.

–f Returns the name of the log file to which Tivoli Inventory status information is logged.

–h Returns the name of the managed node host to which Tivoli Inventory status information is logged.

–l Returns one of the following values, which indicates where Tivoli Inventory logs status information:

NOTICE_GROUP
Sends status information to the Tivoli Inventory notice group.

LOG_FILE Logs status information to the log file specified with **wsetinvglobal –h** and **–l** options.

TEC Sends status information to the Tivoli Enterprise Console (TEC) console.

OFF Logs no status.

–m Returns a value that indicates whether a distribution to a mobile or roaming endpoint (such as a laptop) is hidden. Hidden distributions are not revealed to the roaming user and are automatically distributed to the roaming endpoint. Distributions that are not hidden are visible to the roaming user, and the roaming user can defer the distribution. One of the following values can be returned:

Y The distribution is hidden.

N The distribution is not hidden.

–n Returns information about when notification is sent when a scan completes on each target. The following values can be returned:

IMMEDIATE Indicates that notification is sent as the scan on each target completes.

DONE Indicates that notification is sent only when scans on all targets are complete.

BUNDLE Indicates that notification is sent periodically, based on the settings for the inventory data handler. Refer to the **wcrtinvdh**, **wgetinvdh**, and **wsetinvdh** commands for more information on bundling.

–t Returns information about what type of notification is sent for a target depending on the completion status of a scan. The following values can be returned:

NONE Indicates that notification is not sent.

SUCCESS Indicates that notification is sent for targets on which scans complete successfully.

FAIL Indicates that notification is sent for targets on which scans failed.

ALL Indicates that notification is sent for all targets.

–u Returns information about the data options for this profile. The following values can be returned:

DIFFS Update with differences. Only data that has been added or changed since the last scan is stored in the configuration repository.

REPLACE Replace with current results. Data in the configuration repository is replaced with the data from the scan.

–w Specifies whether the profile distribution will power up a system that has a wake-on-LAN network card installed and enabled. The following values can be returned:

Y The profile distribution will power up the system.

N The profile distribution will not power up the system.

profile_name Specifies the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run this command with no options to return all global property information for an inventory profile.

Examples

The following example returns all information about the global properties of inventory profile TestProfile:

```
wgetinvglobal @InventoryConfig:TestProfile
```

The output is as follows:

```
Distribution option: Distribute configuration file and run scan
(ALL)
Log file pathname:   /usr/logs/TestProfile.log
Log file host:       fuji
Notice location:     NOTICE_GROUP
Notice interval:     DONE
Notice type:         ALL
Data option:         Replace with current results (REPLACE)
```

The following example returns the location to which notification information is sent and the interval at which it is sent:

```
wgetinvglobal -l -n @InventoryConfig:MyProf
```

The output is as follows:

```
Notice location:     NOTICE_GROUP
Notice interval:     DONE
```

See Also

wgetinvpcfiles, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixhw**, **wgetinvunixsw**, **wsetinvglobal**

wgetinvpcfiles

Lists information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvpcfiles *profile_name*

wgetinvpcfiles –d *profile_name*

wgetinvpcfiles –f *profile_name*

wgetinvpcfiles –m *profile_name*

wgetinvpcfiles –s *profile_name*

Description

The **wgetinvpcfiles** command returns information about a specified profile regarding files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints. This command can return the following information about an inventory profile:

- The directories that are included or excluded during a software scan.
- The files or file types that are included or excluded during a software scan.
- The MIF files to be read during the scan.
- The scripts to be run during the scan.

Options

–d	Returns the list of directories to include or exclude during the scan.
–f	Returns the list of files or file types to include or exclude during the scan.
–m	Returns the list of MIF files to be read during the scan.

–s	Returns a list of scripts to be run on targets during the scan.
profile_name	The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run **wgetinvpcfiles** with no options to return all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected by the specified inventory profile.

Examples

The following example returns all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected by the inventory profile MyProfile:

```
wgetinvpcfiles @InventoryConfig:MyProfile
```

The output from this command is as follows:

```
No Include Directories
Exclude Directories:
        */TMP/
        */TEMP/
Include File Types:
        *.EXE
        *.DLL
        *.COM
        *.NLM
No Custom Mif Files
No script to run after scan
```

See Also

wgetinvglobal, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixhw**, **wgetinvunixsw**, **wsetinvpcfiles**

wgetinvpchw

Lists the options that are set for the hardware scan of a PC endpoint.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvpchw *profile_name*

wgetinvpchw –d *profile_name*

wgetinvpchw –t *profile_name*

wgetinvpchw–u *profile_name*

Description

The **wgetinvpchw** command returns the options for PC hardware scans that are configured for an inventory profile. This command can return the following information about a profile:

- Whether the profile runs the hardware scan, sends the scan data to the configuration repository, or both.
- Which hardware components are scanned for.

Options

–d Lists all the hardware components that you can scan for, and indicates whether the inventory profile is configured to scan for each component.

–t Returns a value that indicates whether the Tivoli hardware scanner is enabled. The following values can be returned:

YES	The Tivoli hardware scanner is enabled.
NO	The Tivoli hardware scanner is disabled.

–u Returns a value that indicates whether the data collected during hardware scans of PC systems is sent to the configuration repository. The following values can be returned:

- **YES** Scan data is sent to the configuration repository after the scan completes.
- **NO** Scan data is not sent to the configuration repository. It is stored on the endpoint.

profile_name The inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Examples

The following example returns all information about the options for PC hardware scans that are configured for inventory profile MyProfile:

```
wgetinvpchw @InventoryConfig:MyProfile
```

The output is as follows:

```
Run Tivoli Scanner: YES
Update the hardware scan results in the configuration repository:
YES
Data to scan:
        Processor: YES
        Memory: YES
        Operating System: YES
        Storage: YES
        IP Address: YES
        Network Adapter: YES
        Partition: YES
        PC System Params: YES
        Pointing Device: YES
        Keyboard: YES
        IPX Address: YES
        Video: YES
        Printer: YES
        USB Device: YES
```

The following example indicates whether inventory profile MyProfile is configured to run the Tivoli hardware scanner:

```
wgetinvpchw -t @InventoryConfig:MyProfile
```

The output is as follows:

```
Run Tivoli Scanner: YES
```

See Also

wgetinvglobal, **wgetinvpcfiles**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixhw**, **wgetinvunixsw**, **wsetinvpchw**

wgetinvpcsw

Lists the options for PC software scans that are configured for an inventory profile.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvpcsw *profile_name*

wgetinvpcsw –b *profile_name*

wgetinvpcsw –c *profile_name*

wgetinvpcsw –f *profile_name*

wgetinvpcsw –h *profile_name*

wgetinvpcsw –r *profile_name*

wgetinvpcsw –s *profile_name*

wgetinvpcsw –x *profile_name*

Description

The **wgetinvpcsw** command returns the options for PC software scans that are configured for an inventory profile. This command can return information about whether a profile is configured to perform the following actions:

- Scan for basic file information.
- Generate a checksum value of scanned files.
- Apply the custom filter to scans for basic information.
- Scan for header information.
- Scan the Windows registry for information about installed products.
- Compare scanned files against the list of software signatures.
- Scan executable files only.

Options

–b Returns information about scans for basic file information. The following values can be returned:

- **SCAN** The endpoint is scanned for basic file information, and then the scan data is stored on the endpoint.
- **UPDATE** The endpoint is not scanned. Data from a previous scan for basic file information is collected and sent to the configuration repository.
- **BOTH** The endpoint is scanned for basic file information, and then the scan data is collected and sent to the configuration repository.
- **NO** No scan for basic file information is performed. No data for basic file information is collected.

–c Returns information about the configuration of the checksum options. The following values can be returned:

- **NONE** The scan does not generate checksum values of scanned files.
- **QUICK** The scan collects the Quick checksum value of scanned files.
- **FULL** The scan collects the Full checksum value of scanned files.
- **MD5** The scan collects the MD5 checksum value of scanned files.

–f Returns information about whether the custom filter is applied to scans for basic file information. The following values can be returned:

- **YES** The scan applies the custom filter to scans for basic file information.

NO The scan does not apply the custom filter to scans for basic file information.

–h Returns information about scans for header information. The following values can be returned:

SCAN The endpoint is scanned for header information, and then the scan data is stored on the endpoint.

UPDATE The endpoint is not scanned. Data from a previous scan for header information is collected and sent to the configuration repository.

BOTH The endpoint is scanned for header information, and then the scan data is collected and sent to the configuration repository.

NO No scan for header information is performed. No data for header information is collected.

–r Returns information about scans of the Windows registry for information about installed products. The following values can be returned:

SCAN The Windows registry is scanned for information about installed products, and then the scan data is stored on the endpoint.

UPDATE The endpoint is not scanned. Data from a previous scan for registry information is collected and sent to the configuration repository.

BOTH The Windows registry is scanned for information about installed products, and then the scan data is collected and sent to the configuration repository.

	NO	No scan for registry information is performed. No data for registry information is collected.
–s	Compares scanned files against the list of software signatures. The following values can be returned:	
	SCAN	Scanned files are compared to the list of software signatures. Data for matching files is stored on the endpoint.
	UPDATE	A scan for software signature matching is not run. Data from a previous scan for software signature matching is collected and sent to the configuration repository.
	BOTH	Scanned files are compared to the list of software signatures, and then the data for matching files is collected and sent to the configuration repository.
	NO	No scan for software signature matching is performed. No data from software signature matching is collected.
–x	Scan executable files only. The following values can be returned:	
	YES	Only executable files are scanned.
	NO	The scan is not limited to executable files.
profile_name	The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.	

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run this command with no options to return all information about the options for PC software scans that are configured for the specified inventory profile.

Examples

The following example returns all information about the options for PC software scans that are configured for the inventory profile MyProfile:

```
wgetinvpcsw @InventoryConfig:MyProfile
```

This command returns the following output:

```
Scan for and update basic file information: NO
Cyclic Redundancy Check: No CRC
Scan for and update header information: BOTH
Scan for and update PC registry information: BOTH
Scan for and update matching software signatures: NO
Apply custom filter to basic file information scan results: NO
Executable files only: YES
```

The following example returns the checksum option configured for the inventory profile MyProfile:

```
wgetinvpcsw -c @InventoryConfig:MyProfile
```

This command returns the following output:

```
Cyclic Redundancy Check: No CRC
```

See Also

wgetinvglobal, **wgetinvpcfiles**, **wgetinvpchw**, **wgetinvunixfiles**, **wgetinvunixhw**, **wgetinvunixsw**, **wsetinvpcsw**

wgetinvunixfiles

Lists information about the specified inventory profile regarding files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvunixfiles *profile_name*

wgetinvunixfiles –d *profile_name*

wgetinvunixfiles –f *profile_name*

wgetinvunixfiles –m *profile_name*

wgetinvunixfiles –s *profile_name*

Description

The **wgetinvunixfiles** command returns information files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints. This command can return the following information about an inventory profile:

- The directories that are included or excluded during a software scan.
- The files or file types that are included or excluded during a software scan.
- The MIF files to be read during the scan.
- The scripts to be run during the scan.

Options

–d	Returns the list of directories to include or exclude during the scan.
–f	Returns the list of files or file types to include or exclude during the scan.
–m	Returns the list of MIF files to be read during the scan.

–s	Returns a list of scripts to be run on targets during the scan.
profile_name	The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run this command with no options to return all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected for the specified inventory profile.

Examples

The following example returns all information about files and directories to be scanned, scripts to be run, and custom MIF files to be collected by the inventory profile MyProfile:

```
wgetinvunixfiles @InventoryConfig:MyProfile
```

The output from this command is as follows:

```
No Include Directories
Exclude Directories:
        */tmp/
Exclude File Types:
        *.c
        *.txt
No Custom Mif Files
No script to run after scan
```

See Also

wgetinvglobal, **wgetinvpcfiles**, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixhw**, **wgetinvunixsw**, **wsetinvunixfiles**

wgetinvunixhw

Lists the options that are set for the hardware scan of a UNIX endpoint.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvunixhw *profile_name*

wgetinvunixhw –d *profile_name*

wgetinvunixhw –t *profile_name*

wgetinvunixhw–u *profile_name*

Description

The **wgetinvunixhw** command returns the options for UNIX hardware scans that are configured for an inventory profile. This command can return the following information about a profile:

- Whether the profile runs the hardware scan, sends the scan data to the configuration repository, or both.
- Which hardware components are scanned for.

Options

–d Lists all the hardware components that you can scan for, and indicates whether the inventory profile is configured to scan for each component.

–t Returns a value that indicates whether the Tivoli hardware scanner is enabled. The following values can be returned:

YES	The Tivoli hardware scanner is enabled.
NO	The Tivoli hardware scanner is disabled.

–u Returns a value that indicates whether the data collected during hardware scans of UNIX systems is sent to the

configuration repository. The following values can be returned:

YES	Scan data is sent to the configuration repository after the scan completes.
NO	Scan data is not sent to the configuration repository. It is stored on the endpoint.

profile_name The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Examples

The following example returns all information about the options for UNIX hardware scans that are configured for inventory profile MyProfile:

```
wgetinvunixhw @InventoryConfig:MyProfile
```

The output is as follows:

```
 Run Tivoli Scanner: YES
Update the hardware scan results in the configuration repository:
YES
Data to scan:
        Processor: YES
        Memory: YES
        Operating System: YES
        Storage: YES
        IP Address: YES
        Network Adapter: YES
        Partition: YES
        Pointing Device: YES
        Keyboard: YES
        UNIX System Params: YES
```

The following example indicates whether inventory profile MyProfile is configured to run the Tivoli hardware scanner:

```
wgetinvunixhw -t @InventoryConfig:MyProfile
```

The output is as follows:

```
Run Tivoli Scanner: YES
```

See Also

wgetinvglobal, **wgetinvpcfiles**, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixsw**, **wsetinvunixhw**

wgetinvunixsw

Lists the options for UNIX software scans that are configured for an inventory profile.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetinvunixsw *profile_name*

wgetinvunixsw –b *profile_name*

wgetinvunixsw –c *profile_name*

wgetinvunixsw –f *profile_name*

wgetinvunixsw –p *profile_name*

wgetinvunixsw –s *profile_name*

wgetinvunixsw –x *profile_name*

Description

The **wgetinvunixsw** command returns the options for UNIX software scans that are configured for an inventory profile. This command can return information about whether a profile is configured to perform the following actions:

- Scan for basic file information.
- Generate the checksum value of scanned files.
- Apply the custom filter to scans for basic information.
- Scan the operating system for information about installed products.
- Compare scanned files against the list of software signatures.
- Scan executable files only.

Options

–b Returns information about scans for basic file information. The following values can be returned:

- **SCAN** The endpoint is scanned for basic file information, and then the scan data is stored on the endpoint.
- **UPDATE** The endpoint is not scanned. Data from a previous scan for basic file information is collected and sent to the configuration repository.
- **BOTH** The endpoint is scanned for basic file information, and then the scan data is collected and sent to the configuration repository.
- **NO** No scan for basic file information is performed. No data for basic file information is collected.

–c Returns information about the configuration of the checksum options. The following values can be returned:

- **NONE** The scan does not generate checksum values of scanned files.
- **QUICK** The scan collects the Quick checksum value of scanned files.
- **FULL** The scan collects the Full checksum value of scanned files.
- **MD5** The scan collects the MD5 checksum value of scanned files.

–f Returns information about whether the custom filter is applied to scans for basic file information. The following values can be returned:

- **YES** The scan applies the custom filter to scans for basic file information.

	NO	The scan does not apply the custom filter to scans for basic file information.
–p	Returns information about scans of the operating system for information about installed products. The following values can be returned:	
	SCAN	The operating system is scanned for information about installed products, and then the scan data is stored on the endpoint.
	UPDATE	The endpoint is not scanned. Data from a previous scan for information about installed products is collected and sent to the configuration repository.
	BOTH	The operating system is scanned for information about installed products, and then the scan data is collected and sent to the configuration repository.
	NO	No scan for information about installed products is performed. No data for information about installed products is collected.
–s	Compare scanned files against the list of software signatures. The following values can be returned:	
	SCAN	Scanned files are compared to the list of software signatures. Data for matching files is stored on the endpoint.
	UPDATE	A scan for software signature matching is not run. Data from a previous scan for software signature matching is collected and sent to the configuration repository.
	BOTH	Scanned files are compared to the list of software signatures, and then the

		data for matching files is collected and sent to the configuration repository.
	NO	No scan for software signature matching is performed. No data from software signature matching is collected.
–x	Scan executable files only. The following values can be returned:	
	YES	Only executable files are scanned.
	NO	The scan is not limited to executable files.
profile_name	The name of the inventory profile about which you want information. Enter the profile name in the format **@InventoryConfig:***profile_name*.	

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Notes

Run this command with no options to return all information about the options for UNIX software scans that are configured for the specified inventory profile.

Examples

The following example returns all information about the options for UNIX software scans that are configured for the inventory profile MyProfile:

```
wgetinvunixsw @InventoryConfig:MyProfile
```

This command returns the following output:

```
 Scan for and update basic file information: NO
Cyclic Redundancy Check: No CRC
Scan for and update Unix software package information: BOTH
Scan for and update matching software signatures: NO
Apply custom filter to basic file information scan results: NO
Executable files only: YES
```

The following example returns the checksum option configured for the inventory profile MyProfile:

```
wgetinvunixsw -c @InventoryConfig:MyProfile
```

This command returns the following output:

```
Cyclic Redundancy Check: No CRC
```

See Also

wgetinvglobal, **wgetinvpcfiles**, **wgetinvpchw**, **wgetinvpcsw**, **wgetinvunixfiles**, **wgetinvunixhw**, **wsetinvunixsw**

wgetscanstat

Returns information about current inventory scans.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wgetscanstat

wgetscanstat –a [**–s**] [**–f**] [**–p**]

wgetscanstat –i *id* [**–i** *id*]… [**–s**] [**–f**] [**–p**]

Description

The **wgetscanstat** command returns information about current inventory scans. If you run this command without any options, it returns a list of scan IDs of current inventory scans along with the names of the inventory profiles that initiated those scans. With the available options, you can retrieve information about success, failure, or pending status of all or specific scans.

Options

–a	Returns detailed information about all current scans, including the scan ID, the profile name, the start time, elapsed time, and the number of targets on which scans are completed or pending.
–s	Returns a list of all the targets on which scans have completed successfully.
–f	Returns a list of all the targets on which scans have failed.
–p	Returns a list of all the targets on which scans are still pending.
–i *id*	Returns information about the scan with the specified scan ID. You can specify multiple scan IDs.

Authorization

user, **admin**, **senior**, **super**, or **Inventory_view**

Examples

The following example returns detailed information about all running scans and lists failed, successful, and pending scans.

```
wgetscanstat –a –f –s –p
```

The output is as follows:

```
Scan Id:            144
Profile Name:       Hardware
Start time:         Thu Aug 05 17:18:57 1999
Elapsed time:       0 Days 0 Hours 0 Minutes 11 Seconds
Clients completed:  3
Clients pending:    1
The following clients have successfully completed:
    mckinley
    aztlan–1
The following clients have failed:
    alioth–5
The following clients are still pending:
    suntmp18–3

Scan Id:            145
Profile Name:       Hardware
Start time:         Thu Aug 05 17:18:57 1999
Elapsed time:       0 Days 0 Hours 0 Minutes 11 Seconds
Clients completed:  0
Clients pending:    1
The following clients have successfully completed:
The following clients have failed:
The following clients are still pending:
    suntmp11
```

See Also

wcollect, **wcstat**

winvfilter

Modifies files from the custom filter list in the configuration repository.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

winvfilter –a –f *input_file*

winvfilter –r –f *input_file*

winvfilter –a –n *filter_name*

winvfilter –r –n *filter_name*

winvfilter –a –i

winvfilter –r –i

Description

You can specify that scans for basic information include only files listed in the custom filter. The **winvfilter** command enables you to add or delete file names from the custom filter list. The custom filter list is stored in the configuration repository. You can add one file name at a time using the **–n** option, or you can add multiple file names using the **–f** option. For more information about the custom filter, see the Tivoli Inventory online help.

Note: On UNIX systems, file names are case sensitive.

You enable the custom filter using the **wsetinvpcsw** or **wsetunixsw** command and **–f** option, or by selecting the Apply custom filter option on the Software Scan Configuration window.

Options

–a	Adds the specified file or list of files to the custom filter.
–f *input_file*	Specifies the name of a text file that contains a list of file names to be added to, or removed from, the custom filter. Within the input file, you must list each file name on a separate line.

–i	Specifies filter data to be read from standard input. You enter data by piping input from a command.
–n *filter_name*	Specifies a file name to be added to, or removed from, the custom filter.
–r	Removes the specified file or list of files from the custom filter.

Authorization

RIM_update, user, admin, senior, or **super**

Examples

The following example adds the list of file names from the file filterlist.txt to the custom filter.

```
winvfilter -a -f filterlist.txt
```

The following example removes the file name myapp.exe from the custom filter.

```
winvfilter -r -n myapp.exe
```

The following example loads the file names from the software signature table (SWARE_SIG) into the custom filter list. This example is for a DB2 RDBMS and assumes that the DB2 environment is sourced and a connection to the database has been established:

```
db2 "select sware_name from sware_sig" | winvfilter -a -i
```

See Also

winvsig, **wsetinvpcsw**, **wsetinvunixsw**

winvrmnode

Removes from the configuration repository all scanned information about an endpoint.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

winvrmnode –i *computer_system_ID …*

winvrmnode –l *TME_object_label …*

winvrmnode –o *TME_object_ID …*

Description

The **winvrmnode** command removes hardware and software information for a specified endpoint from the Tivoli Inventory configuration repository. This command does not remove data from history tables or custom tables.

You can specify an endpoint by providing either the name (*TME_object_label*), object ID (*TME_object_ID*), or computer system ID (*computer_system_ID*) of the system. If more than one endpoint has the same name, using *TME_object_label* removes the Tivoli Inventory information for all endpoints with that name, and using *TME_object_ID* or *computer_system_ID* removes the information for just the specified endpoint. You can remove information for more than one endpoint by providing a list of names.

Note: A system's object ID is the value of TME_OBJECT_ID. To find the object ID for a system, run the INVENTORY_HWARE query with the TME_OBJECT_ID and TME_OBJECT_LABEL as chosen columns. The results show system names and their corresponding object IDs. A system's computer system ID is the value of COMPUTER_SYS_ID. To find the computer system ID for a system, run the INVENTORY_HWARE query.

Options

–i *computer_system_ID*

Removes from the configuration repository all hardware and software information for the endpoint with the specified computer system ID.

–l *TME_object_label*

Removes from the configuration repository all hardware and software information for the endpoint with the specified name. If multiple instances of the same label exist, this option removes all instances.

–o *TME_object_ID*

Removes from the configuration repository all hardware and software information for the endpoint with the specified object ID.

Authorization

super

Notes

For information about using **winvrmnode** to remove scan data from custom tables, see "Adding a Custom Table to the Configuration Repository" on page 7-20.

Examples

The following command removes all hardware and software information for the endpoints named nile and tigris.

```
winvrmnode -l nile tigris
```

winvsig

Modifies the list of software signatures in the configuration repository.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

winvsig –a –f *file_name*

winvsig –a –i

winvsig –a –n *name* **–s** *size* **–d** *description* **–v** *version*
[**–q** *quick_checksum* | **–c** *full_checksum* | **–m** *MD5_checksum*]

winvsig –r –f *file_name*

winvsig –r –i

winvsig –r –n *name* **–s** *size*

Description

The **winvsig** command adds, modifies, or removes software signatures from the configuration repository. During a scan that uses software signature matching, Tivoli Inventory matches the files found on the endpoint with software signature data. If the name and size of the scanned file match the file name and size values of a software signature, Tivoli Inventory sends the software signature data for the matched file to the configuration repository.

Note: On UNIX systems, the file name value is case sensitive.

Options

–a Adds one or more software signatures to the configuration repository, and modifies existing software signatures.

–c *full_checksum*
Specifies the Full CRC32 checksum value of the file you enter with the **–n** option. This must be an eight-character hexadecimal value with any alphabetic characters in lowercase.

–d *description* Specifies a name or description for the software that is associated with the file you entered with the **–n** option. If the description includes spaces, enclose it in double quotation marks (" "). This information appears on the Tivoli Inventory GUI in the Software Name column of the Signatures window.

–f *file_name* Specifies the name of a file that contains the data for one or more software signatures. The software signature data in the file must be in the following format:

```
<I>,name,size,description,version,quick_checksum,
full_checksum,md5_checksum
```

The *quick_checksum*, *full_checksum*, and *md5_checksum* values are optional. You can specify one or more of the values. However, you must specify the values in the order shown. Moreover, if you skip one of the values, you must provide a placeholder in the form of two double quotation marks (""). For example, to specify values for *full_checksum* and *md5_checksum* but not *quick_checksum*, enter the following data:

```
<I>,name,size,description,version,"",
full_checksum,md5_checksum
```

–i Specifies software signature data to be read from standard input. You enter data by piping input from a command.

–m *MD5_checksum* Specifies the MD5 checksum value of the file you entered with the **–n** option. This must be a 32-character hexadecimal value with any alphabetic characters in lowercase.

–n *name* Specifies the name of the file that you want to use to identify the software product. This file is usually the primary executable for a software product, for example notepad.exe. The combination of the file name entered with the **–n** option and file size entered with the **–s** option must be unique in the list of software signatures.

Enter a file name that matches the file name exactly, including case. If the file name differs across operating systems, you might need to create a software signature for every operating system. For example, a file named SAMPLE.EXE is not the same as a file named sample.exe.

–q *quick_checksum*
: Specifies the Quick checksum value of the file you entered with the **–n** option. This must be an eight-character hexadecimal value with any alphabetic characters in lowercase.

–r
: Removes one or more software signatures from the configuration repository.

–s *size*
: Specifies the size of the file, in bytes, that you entered with the **–n** option. The combination of the file name entered with the **–n** option and file size entered with the **–s** option must be unique in the list of software signatures.

  Note: You must enter a size greater than zero bytes. The **winvsig** command does not work if you enter 0 for this option.

–v *version*
: Specifies the version of the software package that is associated with the file you entered with the **–n** option.

Authorization

RIM_update, user, admin, senior, or **super**

Notes

For all hexadecimal values that you supply using this command, you must include any zeroes:

Incorrect: `ff1234`

Correct: `00ff1234`

Examples

The following example adds a software signature to the configuration repository for a version of the Windows NT WordPad software package.

```
winvsig –a –n wordpad.exe –s 204800 –d “WordPad” –v 1.0
```

The following example removes from the configuration repository the software signature data listed in a file named files.txt.

```
winvsig –r –f files.txt
```

The following example updates the version for an existing software signature to 1.2 and adds the Quick checksum value.

```
winvsig –a –n sample.exe -s 2048 -v 1.2 -q fd123c34
```

See Also

winvfilter, **wqueryinv**

wqueryinv

Queries the configuration repository for information about a client in the Tivoli environment.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wqueryinv –d *delimiter* {**–o** *ep_oid* | *ep_name*}

wqueryinv [**–h** *host_name*] **–f** *file_name* {**–o** *ep_oid* | *ep_name*}

wqueryinv –n {**–o** *ep_oid* | *ep_name*}

wqueryinv {**–s** | **–q** @*query_name*} {**–o** *ep_oid* | *ep_name*}

Description

The **wqueryinv** command retrieves inventory scan information for an endpoint in the Tivoli environment. Using this command, you can run any query against the configuration repository that includes the TME_OBJECT_LABEL and TME_OBJECT_ID columns for any endpoint in your Tivoli environment. You can send the query results to either standard output or a file. If you do not specify a query, the command will execute the INVENTORY_HWARE query for the specified client.

To use **wqueryinv** to query about an endpoint, you can provide either the endpoint's object ID or host name.

Options

–d *delimiter* Specifies a delimiter for the query results file. Enclose the delimiting character in double quotation marks (" "). If you do not specify a delimiter, a comma will be used.

–f *file_name* Specifies the location and file name in which to save the results of the query. You must provide both a directory and a file name. If you do not use the **–f** option, the query results will be sent to standard output.

–h *host_name* Specifies the name of the managed node on which you want to save the query results file. If you do not use the **–h** option, the query results will be saved to the local system.

–n Indicates that headers should not be included in the query results file.

–o *client_oid* Specifies the Tivoli object ID of the endpoint to be queried.

–q @*query_name*
Specifies the query to be executed.

–s Runs the INVENTORY_SWARE query, which returns software information for the specified client. If you do not use the **–s** or **–q** options, the INVENTORY_HWARE query will be executed.

ep_name Specifies the host name of the endpoint to be queried.

Authorization

RIM_view, **Query_view**, **Query_execute**, **Query_edit**, **senior**, or **super**

Examples

The following example runs the query HDISK_QUERY for an endpoint named alister, and sends the output to a file named /tmp/inv.out on a system named mckinley.

```
wqueryinv -n -d ";" -h mckinley -f /tmp/inv.out \
-q @HDISK_QUERY alister
```

See Also

wrunquery (in the *Tivoli Management Framework Reference Manual*)

wsetinvdh

Changes the settings of the inventory data handler.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvdh [**–d** *status_directory*]

wsetinvdh [**–n** *bundle_every_n_minutes*]

wsetinvdh [**–q** *bundle_every_n_targets*]

wsetinvdh [**–r** *max_RDBMS_retries*]

wsetinvdh [**–s** {**YES** | **NO**}]

wsetinvdh [**–t** *RDBMS_retry_delay_time*]

Description

The **wsetinvdh** command allows you to make changes to the inventory data handler that you created using the **wcrtinvdh** command. The options are identical to the **wcrtinvdh** command except that you do not specify a managed node. The **wsetinvdh** command automatically makes the changes to the inventory data handler instance @InvDataHandler:inv_data_handler in the Tivoli object database.

Options

–d *status_directory*

Specifies where the inventory data handler stores status information that can be restored in case of a system failure. By default, the location is $DBDIR/inventory/stat_dir on the managed node where the inventory data handler resides.

–n *bundle_every_n_minutes*

Specifies the interval at which scan completion notifications are sent (when the **–n** option for the **wsetinvglobal** option is set to **BUNDLE**). If you set this number, Tivoli Inventory sends a notice with a list of the targets on which scans have completed during the specified time period. The default value is 10 minutes.

If no scans complete in the specified time period, no notification is sent.

If you set this option to 0, notification occurs according to the value you set for the **–q** option of the **wcrtinvdh** or **wsetinvdh** command.

–q *bundle_every_n_targets*

Specifies the maximum number of targets in a bundle. A bundle is a group of targets about which status information is sent at one time. Status refers to the success or failure of a scan for each target of a particular inventory profile.

The default value is 10 targets.

This option, in combination with the **–n** option of the **wsetinvglobal** command, configures how status information is sent in bundles.

Notes

- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to 0, no bundling occurs, and you are not notified until the scans on all targets are completed.
- If you set both **–q** *bundle_every_n_targets* and **–n** *bundle_every_n_minutes* to a positive value, bundling occurs when either value (the specified number of targets or minutes) is reached.

–r *max_RDBMS_retries*

Specifies the number of times the inventory data handler tries to write data to the RDBMS server. After the maximum number of retries is reached, a failure notification is sent.

The default value is 5.

–s *option* Specifies whether the inventory data handler stores status information that can be restored in case of a system failure. Use one of the following options:

YES	Status information is stored. This is the default option.
NO	Status information is not stored.

–t *RDBMS_retry_delay_time*

Specifies a value from which to calculate the timeout period in seconds between retries of writes to a RIM object. This timeout period works according to the algorithm *timeout*retry_count*. For example, on the first retry, with a timeout value of 30 seconds, the algorithm sets the timer to 30 * 1 or 30 seconds. On the second retry, the timer sets to 30 * 2 or 1 minute.

The default value is 30 seconds.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following example changes the maximum number of targets in a notification bundle to nine and the RIM object retry delay time to 20 seconds.

```
wsetinvdh –q 9 –t 20
```

See Also

wcollect, **wcrtinvdh**, **wgetinvdh**, **wsetinvglobal**

wsetinvglobal

Sets the global properties for an inventory profile.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvglobal –d {**CONFIG** | **ALL**} *profile_name*

wsetinvglobal –e *seconds profile_name*

wsetinvglobal –f *log_file_name profile_name*

wsetinvglobal –h *log_file_host profile_name*

wsetinvglobal –l {**OFF** | *option_list*} *profile_name*

wsetinvglobal –m {**Y** | **N**} *profile_name*

wsetinvglobal –n {**IMMEDIATE** | **BUNDLE** | **DONE**} *profile_name*

wsetinvglobal –t {**SUCCESS** | **FAIL** | **ALL**} *profile_name*

wsetinvglobal –u {**REPLACE** | **DIFFS**} *profile_name*

wsetinvglobal –w {**Y** | **N**} *profile_name*

Description

The **wsetinvglobal** command enables you to set the global properties for an inventory profile. This command sets the following properties:

- Distribution options—whether the inventory profile only distributes the configuration file, or distributes the configuration file and then runs the scan.
- Data options—whether the scan sends only new or changed data to the configuration repository, or replaces existing data in the configuration repository with the data gathered by the scan.
- Status options—where and under what conditions scan status notification is sent. You can also specify the name and path of the status log file and the managed node on which it is stored.

Options

–d *option* Specifies the distribution option. Use one of the following options:

CONFIG Distributes the configuration file only.

ALL Distributes the configuration file and runs the scan. This is the default option.

–e *seconds* Specifies the period, in seconds, that a profile attempts to scan an endpoint before it times out. The default value is 1800 seconds.

–f *log_file_name*

Specifies the location and name of a file to which you want to log Tivoli Inventory status information. The path must be a full path, not a relative path. If a null string ("") is specified, the log will be written to a file named *$TMPDIR*/inv_scan_*nn*.log where *nn* is the scan ID of the scan and *$TMPDIR* is a temporary directory. The temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows NT: c:\temp

–h *log_file_host*

Specifies the name of a managed node to which you want to save Tivoli Inventory status information. If a null string ("") is specified, the log file host will be the managed node on which the inventory data handler was created.

–l {OFF | *option_list***}**

Specifies the location or locations to which notifications are sent. Use one or more of the following options:

NOTICE_GROUP

Sends status information to the Tivoli Inventory notice group.

LOG_FILE Sends status information to the log file specified with the **–f** and **–h** options.

TEC Sends information to the TEC console.

OFF Sends no status.

The default value is NOTICE_GROUP. You can write to more than one location by specifying multiple options. Separate the options with a comma (,).

Note: If an error occurs when Tivoli Inventory tries to write to a log file, an error message is sent to the Tivoli Inventory notice group and all status information is written only to the Tivoli Inventory notice group.

–m Specifies whether a distribution to a mobile or roaming endpoint (such as a laptop) is hidden. Hidden distributions are not revealed to the roaming user and are automatically distributed to the roaming endpoint. Distributions that are not hidden are visible to the roaming user, and the roaming user can defer the distribution. Specify one of the following options:

Y The distribution is hidden.

N The distribution is not hidden.

–n *option* Specifies when notification is sent when a scan completes on each target. Choose one of the following options:

IMMEDIATE Notification is sent when the scan on each target completes.

BUNDLE Notification is sent periodically, based on the settings for the inventory data handler. Refer to the **wcrtinvdh**, **wgetinvdh**, and **wsetinvdh** commands for more information on bundling.

DONE Notification is sent only when scans on all targets are complete.

The default value is DONE.

–t *option* Sets the circumstances under which a notice is sent for the specified profile. Specify one of the following options:

NONE	Notification is not sent.
SUCCESS	Notification is sent for targets on which scans complete successfully.
FAIL	Notification is sent for targets on which scans failed.
ALL	Notification is sent for all targets.

The default value is ALL.

–u *option* Specifies the data option. Use one of the following options:

REPLACE	Replaces existing data in the configuration repository with the data gathered by this scan.
DIFFS	Sends to the configuration repository only data that has been added or changed since the last scan. Data from previous scans that is not present in the current scan is deleted from the configuration repository. No record of changes is kept unless history tables are used. This is the default option.

–w Specifies whether the profile distribution will automatically power up a system that has a wake-on-LAN network card installed and enabled. Specify one of the following options:

Y	The profile distribution will power up the system.
N	The profile distribution will not power up the system.

profile_name The profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following example configures the inventory profile SW_Profile to replace existing data in the configuration repository with the data gathered by the scan and send status information as each target completes.

```
wsetinvglobal -u REPLACE -n IMMEDIATE @InventoryConfig:SW_Profile
```

The following example configures the inventory profile SW_Profile to send notification to the log file, the Tivoli Inventory notice group, and the TEC console.

```
wsetinvglobal -l LOG_FILE,NOTICE_GROUP,TEC \
@InventoryConfig:SW_Profile
```

See Also

wgetinvglobal, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvpcsw**, **wsetinvunixfiles**, **wsetinvunixhw**, **wsetinvunixsw**

wsetinvpcfiles

Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvpcfiles –e {**INCLUDE** | **EXCLUDE**} *profile_name*

wsetinvpcfiles –f {**+** | **–**}*file_type profile_name*

wsetinvpcfiles –m {**+** | **–**}*MIF_path profile_name*

wsetinvpcfiles –s *script_path profile_name*

wsetinvpcfiles –t {**INCLUDE** | **EXCLUDE**} **–d** {**+** | **–**}*directory profile_name*

Description

The **wsetinvpcfiles** command specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of PC endpoints. Using this command, you can set the following options:

- Files or file types to include or exclude during the scan.
- Directories to include or exclude during the scan.
- Scripts to be run during the profile distribution after the scan completes.
- Custom MIF files to be collected during the profile distribution after the scan completes.

Options

–d *option directory*

Specifies a directory that you want to add to, or remove from, the list of directories to include or exclude during the scan. By default, the */tmp and */temp directories are excluded from scans. You must precede this option with the **–t** option to indicate whether the directory is to be included or excluded during the scan. Use one of the

following options. Do not type a space between the option and the value it modifies.

+ Adds this directory to the list.

– Removes this directory from the list.

When specifying a directory to *include* in a scan, observe the following guidelines:

- You cannot use wildcard characters. All characters are treated as literals, including asterisks (*) and question marks (?).
- If you do not specify a drive (for example c:/ or sys:/), Tivoli Inventory scans the specified directory on all scanned drives.
- All directories are considered absolute paths. For example, if you specify abc, c:/abc will be scanned, but c:/temp/abc will not.

When specifying a directory to *exclude* from a scan, observe the following guidelines:

- You can use an asterisk as a wildcard at the beginning or end of a path name, for example *abc, */abc, or abc/def*.
- You can use an asterisk as a wildcard after a drive designation, for example c:*/abc, or sys:*/abc.
- You cannot use a wildcard within a path name, for example /*/temp, c:/*/temp, te*mp, or /user*/temp). All characters within paths are treated as literals.
- You cannot use a wildcard in place of a drive letter.
- You must enclose directories that have a wildcard asterisk character in double quotation marks (" ").

–e *option*
Specifies whether the list of files or file types created with the **–f** option is included or excluded during the scan. Use one of the following options:

INCLUDE	Includes the files or file types in the scan. This is the default option.
EXCLUDE	Excludes the files or file types from the scan.

–f *option file_type*
Specifies a file or file type that you want to add to, or remove from, the list of files or file types to include or exclude during the scan. You can specify a file by typing the file name, or a file type by using an asterisk (*) as a wildcard. You can place the asterisk at the beginning or end of the text string, or both (for example *.bat, abc.*, *abc.bat, abc.ba*, *abc, abc*, *abc*). You cannot place the asterisk within the text string (for example ab*c, ab*c.bat, abc.*at). You must enclose file types in double quotation marks (" "). You can repeat the **–f** option multiple times to add or remove multiple files or file types. Use one of the following options. Do not type a space between the option and the value it modifies.

+	Adds the file or file type to the list.
–	Removes the file or file type from the list.

–m *option MIF_path*
Specifies the path and name of a MIF file that you want to add to, or remove from, the list of MIF files to collect during the profile distribution. For *MIF_path*, you must specify the full path and name of the MIF file. You can repeat the **–m** option multiple times to add or remove multiple MIF files. Use one of the following options.

Do not type a space between the option and the value it modifies.

+ Add the MIF file to the list.

– Remove the MIF file from the list.

–s *script_path* Path and name of a script that you want to run on an endpoint during the profile distribution. The script runs after the scan completes and before MIF files are read. You can specify text files only. You cannot specify binary files.

Notes

- If you edit a script after adding it, you must then add the script again using the **–s** option. If you do not, the revised script will not be run.
- To remove a script, specify double quotation marks with no characters in between ("").

–t *option* Specifies whether the directory specified with the **–d** option that follows this option is to be included or excluded during software scans of PC systems. Use one of the following options:

INCLUDE Includes the specified directory in the scan.

EXCLUDE Excludes the specified directory from the scan.

profile_name Specifies the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Notes

Using the **–t** and **–d** options, you can create separate lists of directories to be excluded and directories to be included. However, you cannot do this with the list of files or file types (the **–f** option). For files and file

types, you use the **–e** option to specify whether the entire list will be either included or excluded during a scan.

Examples

The following example adds the /cache directory to the list of directories to exclude during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -t EXCLUDE -d +/cache @InventoryConfig:SW_Profile
```

The following example removes the /tmp and /temp directories from the list of directories to exclude during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -t EXCLUDE -d -"*/tmp" -d-"*/temp" \
@InventoryConfig:SW_Profile
```

The following example removes the /usr directory from the list of directories to include during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -t INCLUDE -d -/usr @InventoryConfig:SW_Profile
```

The following example specifies that the list of files and file types specified by the **–f** option should be excluded from the scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -e EXCLUDE @InventoryConfig:SW_Profile
```

The following example adds the .bat and .sys file types to the files to be filtered during the scan. These files are either included or excluded during the scan depending on the setting of the **–e** option. This change is applied to an inventory profile named SW_Profile.

```
wsetinvpcfiles -f +"*.bat" –f +"*.sys" @InventoryConfig:SW_Profile
```

The following example specifies that the file TEST.BAT is run during the profile distribution. This change is applied to an inventory profile named SW_Profile.

Note: Because the path for TEST.BAT is not specified in this example, TEST.BAT must be in the directory from which this command is run.

```
wsetinvpcfiles -s "TEST.BAT" @InventoryConfig:SW_Profile
```

See Also

wgetinvpcfiles, **wsetinvglobal**, **wsetinvpchw**, **wsetinvpcsw**, **wsetinvunixfiles**, **wsetinvunixhw**, **wsetinvunixsw**

wsetinvpchw

Sets the options for the hardware scan of PC endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvpchw –a *component profile_name*

wsetinvpchw –r *component profile_name*

wsetinvpchw –t {**Y** | **N**} *profile_name*

wsetinvpchw –u {**Y** | **N**} *profile_name*

Description

The **wsetinvpchw** command sets the options for the hardware scan of a PC endpoint. You can specify the following options:

- Whether to run the Tivoli hardware scanner during scans of PC endpoints.
- Whether the scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components you want to scan for.

Options

–a *component*

Adds the specified component to the list of hardware components to scan for. You can specify the following components.

Processor Describes the processor or processors installed on the system.

OperatingSystem
Describes only the operating system running at the time of the scan. This option does not list all installed operating systems on dual-boot systems.

IPAddress	Lists the TCP/IP configuration information.
Partition	Describes each of the drive partitions on a system. A system might have only one partition. General information listed can include the drive mapping, the size and available size of each partition, the file system, and the media type.
Keyboard	Describes the connected keyboard, if any.
IPXAddress	Lists IPX configuration information.
Printer	Describes any local and network printers connected to the system.
Memory	Describes the installed memory. This option can list details such as the amount of actual memory, virtual memory, and paging space.
Storage	Describes all local storage devices such as hard-disk drives, CD-ROM drives, and removable-media drives. This option does not list network drives.
NetworkAdapter	Describes the installed network adapter or adapters.
PointingDev	Describes the type of pointing device (such as a mouse or trackball) attached to the system.
Video	Describes the video adapter and the current video settings. This option does not list the maximum possible settings.
USBDev	Describes all USB devices and hubs attached to the system at the time of the scan.

PCI Lists the expansion cards installed on the PCI and AGP buses.

PCSystemParams
Lists PC-specific information such as user name, domain name, and BIOS characteristics.

Note: These component values are not case sensitive. Moreover, you do not need to enter the entire component name. You can enter only the number of characters necessary to uniquely identify that component. You can specify multiple components by repeating **–a** *component* multiple times.

–r *component*

Removes the specified component from the list of hardware components to scan for. The possible values for *component* are the same as for the **–a** option. You can specify multiple components by repeating **–r** *component* multiple times.

–t *option* Specifies whether to run the Tivoli hardware scanner. Use one of the following options:

Y Runs the Tivoli hardware scanner.

N Does not run the Tivoli hardware scanner.

–u *option* Specifies whether to update the configuration repository with the scan data. Use one of the following options:

Y Runs the scan and send the scan data to the configuration repository.

N Does not send the scan data to the configuration repository. Runs the scan and then stores the scan data on the endpoint.

profile_name

The profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following command configures the inventory profile HW_Profile to run the Tivoli hardware scanner, exclude processor information from the scan, and include memory and drive partition information in the scan.

```
wsetinvpchw -r Processor -a Memory -a Partition -t Y \
@InventoryConfig:HW_Profile
```

See Also

wgetinvpchw, **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpcsw**, **wsetinvunixfiles**, **wsetinvunixhw**, **wsetinvunixsw**

wsetinvpcsw

Sets the options for software scans of PC endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvpcsw –b {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvpcsw –c {**QUICK | FULL | MD5 | NONE**} *profile_name*

wsetinvpcsw –f {**Y | N**} *profile_name*

wsetinvpcsw –h {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvpcsw –r {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvpcsw –s {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvpcsw –x {**Y | N**} *profile_name*

Description

The **wsetinvpcsw** command sets the options for software scans of PC endpoints. You can specify whether to scan the files on a PC endpoint, scan the Windows registry for information about installed programs, or both.

For file scans, you can configure whether the scan performs the following actions:

- Scans for software signature data.
- Scans for basic file information.
- Applies a custom filter to scans for basic file information.
- Scans files for header information.
- Generates checksum values for scanned files.
- Sends scan data to the configuration repository.
- Scans executable files only.

For Windows registry scans, you can configure whether to send scan data to the configuration repository.

Options

–b *option* Specifies whether to scan for basic file information. This type of scan collects the name and path of all scanned files. Other information, such as date of creation, modification, and last access, is collected on some platforms. Use one of the following options:

SCAN	Scans for basic file information, and then stores the scan data on the endpoint.
UPDATE	Does not scan. Sends the data from the last scan for basic file information to the configuration repository.
BOTH	Scans for basic file information, and then sends the scan data to the configuration repository.
NO	Does not scan for basic file information, and does not send data to the configuration repository.

–c *option* Specifies whether to generate a checksum value for scanned files. Use one of the following options:

QUICK	Generates the Quick checksum value. This option produces a 32-bit value based only on the first 1 KB of each file. This is the quickest checksum option, but produces the least reliable value.
FULL	Generates the Full checksum value. This option produces a 32-bit value based on the full contents of each file. This option takes longer than the Quick option, but produces a more reliable value.
MD5	Generates the MD5 checksum value. This option produces a 128-bit value based on the full contents of each file.

The MD5 option provides a more reliable value than the Quick or Full options, but takes longer to complete.

NONE Does not generate checksum values.

–f *option* Specifies whether to filter the scan for basic file information using a custom list of files. You create this list of files using the Custom Filter window or **winvfilter** command. When this option is set to Y, a scan for basic file information includes only the files specified in the custom filter. Use one of the following options:

Y Filters scans for basic file information using the custom filter.

N Does not filter scans for basic file information using the custom filter.

–h *option* Specifies whether to search the header of scanned files for the company name, product name, and product version. This option has no effect on NetWare and OS/2 systems. Use one of the following options:

SCAN Scans for header information, and then stores the scan data on the endpoint.

UPDATE Does not scan. Sends the data from the last scan for header information to the configuration repository.

BOTH Scans for header information, and then sends the scan data to the configuration repository.

NO Does not scan for header information, and does not send data to the configuration repository.

–r *option* Specifies whether to scan the Windows registry for information about installed products. This option has no

effect on NetWare and OS/2 systems. Use one of the following options:

SCAN Scans the Windows registry, and then stores the scan data on the endpoint.

UPDATE Does not scan. Sends the data from the last Windows registry scan to the configuration repository.

BOTH Scans the Windows registry, and then sends the scan data to the configuration repository.

NO Does not scan the Windows registry, and does not send data to the configuration repository.

–s Specifies whether to scan using software signatures. In this type of scan, the name and size of each scanned file is compared against the list of software signatures. Use one of the following options:

SCAN Scans using software signatures, and then stores the scan data on the endpoint.

UPDATE Does not scan. Sends the data from the last software signature scan to the configuration repository.

BOTH Scans using software signatures, and then sends the scan data to the configuration repository.

NO Does not scan using software signatures, and does not send data to the configuration repository.

–x *option* Specifies whether to filter a file scan by scanning for executable files only. For PC systems, an executable file is defined as a file with one of the following

extensions: .EXE, .COM, .CMD, .BAT. Use one of the following options:

Y	Scans for executable files only.
N	Does not scan only for executable files.

profile_name Specifies the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following command configures the inventory profile SW_Profile to generate an MD5 checksum of scanned files, scan the Windows registry, and send the registry scan data to the configuration repository.

```
wsetinvpcsw -c MD5 -r BOTH @InventoryConfig:SW_Profile
```

See Also

wgetinvpcsw, **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvunixfiles**, **wsetinvunixhw**, **wsetinvunixsw**

wsetinvunixfiles

Specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvunixfiles –e {**INCLUDE** | **EXCLUDE**} *profile_name*

wsetinvunixfiles –f {**+** | **–**}*file_type profile_name*

wsetinvunixfiles –m {**+** | **–**}*MIF_path profile_name*

wsetinvunixfiles –s *script_path profile_name*

wsetinvunixfiles –t {**INCLUDE** | **EXCLUDE**} **–d** {**+** | **–**}*directory profile_name*

Description

The **wsetinvunixfiles** command specifies options for files and directories to be scanned, scripts to be run, and custom MIF files to be collected during scans of UNIX endpoints. Using this command, you can set the following options:

- Files or file types to include or exclude during the scan.
- Directories to include or exclude during the scan.
- Scripts to be run during the profile distribution after the scan completes.
- Custom MIF files to be collected during the profile distribution after the scan completes.

Options

–d *option directory*

Specifies a directory that you want to add to, or remove from, the list of directories to include or exclude during the scan. By default, the */tmp directory is excluded from scans. You must precede this option with the **–t** option to indicate whether the directory is to be included or excluded during the scan. Use one of the

following options. Do not type a space between the option and the value it modifies.

+ Adds this directory to the list.

– Removes this directory from the list.

When specifying a directory to *include* in a scan, observe the following guidelines:

- You cannot use wildcard characters. All characters are treated as literals, including asterisks (*) and question marks (?).
- All directories are considered absolute paths. For example, if you specify abc, /abc will be scanned, but /tmp/abc will not.

When specifying a directory to *exclude* from a scan, observe the following guidelines:

- You can use an asterisk as a wildcard at the beginning or end of a path name, for example *abc, */abc, or abc/def*.
- You cannot use a wildcard within a path name, for example /*/tmp, t*mp, or /usr*/tmp). All characters within paths are treated as literals.
- You must enclose directories that have an asterisk character (*) in double quotation marks (" ").

–e *option* Specifies whether the list of files or file types created with the **–f** option is included or excluded during the scan. Use one of the following options:

INCLUDE Includes the files or file types in the scan. This is the default option.

EXCLUDE Excludes the files or file types from the scan.

–f *option file_type*

Specifies a file or file type that you want to add to, or remove from, the list of files or file types to include or exclude during the scan. You can specify a file by

typing the file name, or a file type by using an asterisk (*) as a wildcard. You can place the asterisk at the beginning or end of the text string, or both (for example *.txt, abc.*, *abc.txt, abc.tx*, *abc, abc*, *abc*). You cannot place the asterisk within the text string (for example ab*c, ab*c.txt, abc.t*t). You must enclose file types in double quotation marks (" "). You can repeat the **–f** option multiple times to add or remove multiple file types. Use one of the following options. Do not type a space between the option and the value it modifies.

+ Adds the file or file type to the list.

– Removes the file or file type from the list.

–m *option MIF_path*

Specifies the location and name of a MIF file that you want to add to, or remove from, the list of MIF files to collect during the profile distribution. For *MIF_path*, you must specify the full path and name of the MIF file. You can repeat the **–m** option multiple times to add or remove multiple MIF files. Use one of the following options. Do not type a space between the option and the value it modifies.

+ Adds the MIF file to the list.

– Removes the MIF file from the list.

–s *script_path* Specifies the location and name of a script that you want to run on an endpoint during the profile distribution. The script runs after the scan completes and before MIF files are read. You can specify text files only. You cannot specify binary files.

Notes

- If you edit a script after adding it, you must then add the script again using the **–s** option. If you do not, the revised script will not be run.
- To remove a script, specify double quotation marks with no characters in between ("").

–t *option* Specifies whether the directory specified with the **–d** option that follows this option is to be included or excluded during software scans of UNIX systems. Use one of the following options:

- **INCLUDE** Includes the specified directory in the scan.
- **EXCLUDE** Excludes the specified directory from the scan.

profile_name The name of the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Notes

Using the **–t** and **–d** options, you can create separate lists of directories to be excluded and directories to be included. However, you cannot do this with the list of files or file types (the **–f** option). For files and file types, you use the **–e** option to specify whether the entire list will be either included or excluded during a scan.

Examples

The following example adds the /cache directory to the list of directories to exclude during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvunixfiles -t EXCLUDE -d +/cache @InventoryConfig:SW_Profile
```

The following example removes the /tmp directory from the list of directories to exclude during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvunixfiles -t EXCLUDE -d -"*/tmp" \
@InventoryConfig:SW_Profile
```

The following example removes the /usr directory from the list of directories to include during a software scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvunixfiles -t INCLUDE -d -/usr @InventoryConfig:SW_Profile
```

The following example specifies that the list of files and file types specified by the **–f** option should be excluded from the scan. This change is applied to an inventory profile named SW_Profile.

```
wsetinvunixfiles -e EXCLUDE @InventoryConfig:SW_Profile
```

The following example adds the .txt and .gif file types to the files to be filtered during the scan. These files are either included or excluded during the scan depending on the setting of the **–e** option. This change is applied to an inventory profile named SW_Profile.

```
wsetinvunixfiles -f +"*.txt" +"*.gif" @InventoryConfig:SW_Profile
```

The following example specifies that the script create_mif.sh is run during the profile distribution. This change is applied to an inventory profile named SW_Profile.

Note: Because the path for create_mif.sh is not specified in this example, create_mif.sh must be in the directory from which this command is run.

```
wsetinvunixfiles -s "create_mif.sh" @InventoryConfig:SW_Profile
```

See Also

wgetinvunixfiles, **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvpcsw**, **wsetinvunixhw**, **wsetinvunixsw**

wsetinvunixhw

Sets the options for the hardware scan of UNIX endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvunixhw –a *component profile_name*

wsetinvunixhw –r *component profile_name*

wsetinvunixhw –t {**Y** | **N**} *profile_name*

wsetinvunixhw –u {**Y** | **N**} *profile_name*

Description

The **wsetinvunixhw** command sets the options for the hardware scan of a UNIX endpoint. You can specify the following options:

- Whether to run the Tivoli hardware scanner during scans of UNIX endpoints.
- Whether the scan data is sent to the configuration repository or stored on the endpoint to be collected later.
- Which hardware components you want to scan for.

Options

–a *component*

Adds the specified component to the list of hardware components to scan for. You can specify the following components.

Processor Describes the processor or processors installed on the system.

OperatingSystem
Describes only the operating system running at the time of the scan. This option does not list all installed operating systems on dual-boot systems.

IPAddress Lists the TCP/IP configuration information.

Partition Describes each of the drive partitions on a system. A system might have only one partition. General information listed can include the drive mapping, the size and available size of each partition, the file system, and the media type.

Keyboard Describes the connected keyboard, if any.

Memory Describes the installed memory. This option can list details such as the amount of actual memory, virtual memory, and paging space.

Storage Describes all local storage devices such as hard-disk drives, CD-ROM drives, and removable-media drives. This option does not list network drives.

NetworkAdapter
Describes the installed network adapter or adapters.

PointingDev Describes the type of pointing device (such as a mouse or trackball) attached to the system.

UnixSystemParam
Describes the current state of the system, such as boot time and run level.

Note: These component values are not case sensitive. Moreover, you do not need to enter the entire component name. You can enter only the number of characters necessary to uniquely identify that component. You can specify

multiple components by repeating **–a** *component* multiple times.

–r *component*

Removes the specified component from the list of hardware components to scan for. The possible values for *component* are the same as for the **–a** option. You can specify multiple components by repeating **–r** *component* multiple times.

–t *option* Specifies whether to run the Tivoli hardware scanner. Use one of the following options:

Y	Runs the Tivoli hardware scanner.
N	Does not run the Tivoli hardware scanner.

–u *option* Specifies whether to update the configuration repository with the scan data. Use one of the following options:

Y	Runs the scan and sends the scan data to the configuration repository.
N	Does not send the scan data to the configuration repository. Runs the scan and then stores the scan data on the endpoint.

profile_name The name of the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following command configures the inventory profile HW_Profile to run the Tivoli hardware scanner, include UNIXSystemParam in the scan, and store the scan data on the endpoint.

```
wsetinvunixhw -a UNIXSystemParam -t Y -u N \
@InventoryConfig:HW_Profile
```

See Also

wgetinvunixhw, **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvpcsw**, **wsetinvunixfiles**, **wsetinvunixsw**

wsetinvunixsw

Sets the options for software scans of UNIX endpoints.

Syntax

Each command option is shown on a separate line unless it is required that you use the options together.

wsetinvunixsw –b {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvunixsw –c {**QUICK | FULL | MD5 | NONE**} *profile_name*

wsetinvunixsw –f {**Y | N**} *profile_name*

wsetinvunixsw –p {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvunixsw –s {**SCAN | UPDATE | BOTH | NO**} *profile_name*

wsetinvunixsw –x {**Y | N**} *profile_name*

Description

The **wsetinvunixsw** command sets the options for software scans of UNIX endpoints. You can specify whether to scan the files on a UNIX endpoint, scan the operating system for information about installed programs, or both.

For file scans, you can configure whether the scan performs the following actions:

- Scans for software signature data.
- Scans for basic file information.
- Applies a custom filter to scans for basic file information.
- Generates checksum values for scanned files.
- Sends scan data to the configuration repository.
- Scans executable files only.

For operating system scans, you can configure whether to send scan data to the configuration repository.

Options

–b *option* Specifies whether to scan for basic file information. This type of scan collects the name and path of all scanned files. Other information, such as date of

creation, modification, and last access, is collected on some platforms. Use one of the following options:

SCAN Scans for basic file information, and then stores the scan data on the endpoint.

UPDATE Does not scan. Sends the data from the last scan for basic file information to the configuration repository.

BOTH Scans for basic file information, and then sends the scan data to the configuration repository.

NO Does not scan for basic file information, and does not send data to the configuration repository.

–c *option* Specifies whether to generate a checksum value for scanned files. Use one of the following options:

QUICK Generates the Quick checksum value. This option produces a 32-bit value based only on the first 1 KB of each file. This is the quickest checksum option, but produces the least reliable value.

FULL Generates the Full checksum value. This option produces a 32-bit value based on the full contents of each file. This option takes longer than the Quick option, but produces a more reliable value.

MD5 Generates the MD5 checksum value. This option produces a 128-bit value based on the full contents of each file. The MD5 option provides a more reliable value than the Quick or Full options, but takes longer to complete.

NONE Does not generate checksum values.

–f *option* Specifies whether to filter the scan for basic file information using a custom list of files. You create this list of files using the Custom Filter window or **winvfilter** command. When this option is set to Y, a scan for basic file information includes only the files specified in the custom filter. Use one of the following options:

Y	Filters scans for basic file information using the custom filter.
N	Does not filter scans for basic file information using the custom filter.

–p *option* Specifies whether to scan the operating system for information about installed products and patches. Use one of the following options:

SCAN	Scans the operating system, and then stores the scan data on the endpoint.
UPDATE	Does not scan. Sends the data from the last operating system scan to the configuration repository.
BOTH	Scans the operating system, and then sends the scan data to the configuration repository.
NO	Does not scan the operating system, and does not send data to the configuration repository.

–s Specifies whether to scan using software signatures. In this type of scan, the name and size of each scanned file is compared against the list of software signatures. Use one of the following options:

SCAN	Scans using software signatures, and then stores the scan data on the endpoint.
UPDATE	Does not scan. Sends the data from the last software signature scan to the configuration repository.

	BOTH	Scans using software signatures, and then sends the scan data to the configuration repository.
	NO	Does not scan using software signatures, and does not send data to the configuration repository.
–x *option*	Specifies whether to filter a file scan by scanning for executable files only. For UNIX systems, an executable file is defined as a file that has the execute permission set. Use one of the following options:	
	Y	Scans for executable files only.
	N	Does not scan only for executable files.
profile_name	The name of the inventory profile that you want to edit. Enter the profile name in the format **@InventoryConfig:***profile_name*.	

Authorization

admin, **senior**, **super**, or **Inventory_edit**

Examples

The following command configures the inventory profile SW_Profile to generate a Quick checksum of scanned files, scan the operating system for information about installed products and patches, and send the operating system scan data to the configuration repository.

```
wsetinvunixsw -c QUICK -p BOTH @InventoryConfig:SW_Profile
```

See Also

wgetinvunixsw, **wsetinvglobal**, **wsetinvpcfiles**, **wsetinvpchw**, **wsetinvunixfiles**, **wsetinvunixhw**, **wsetinvpcsw**

C

Configuration Repository

This appendix lists and describes each view and table included in the Tivoli Inventory, Version 4.0, configuration repository schema. This appendix also lists and describes the queries that you can install when you install Tivoli Inventory, Version 4.0. When you install Tivoli Inventory, you also create the Tivoli Inventory configuration repository in the RDBMS. The configuration repository stores information collected during inventory scans.

Note: Information provided in this appendix regarding the table and view definitions of the Tivoli Inventory configuration repository could also be extracted from the system catalogs of your RDBMS. For more information, consult your RDBMS manuals or your database administrator.

For a graphical representation of the tables in the Tivoli Inventory configuration repository, see the *Tivoli Inventory Configuration Repository* poster enclosed with this documentation, or contact your Tivoli customer support representative for a copy. You can also create your own tables and views.

See Chapter 7, "Collecting Custom Information with Tivoli Inventory" for instructions for adding tables to the configuration repository. See Chapter 8, "Querying Inventory Information" for instructions on creating queries. See Appendix B, "Commands" for more information about deleting information from the configuration repository using the **winvrmnode** command.

History Tracking

Tivoli Inventory, Version 3.6.*x*, stored data only from the current scan and the last scan. However, Tivoli Inventory, Version 4.0, provides an optional history tracking feature to store data and change information from all previous scans. Data from the current scan is stored in operational data tables. Operational data tables are overwritten or updated during each scan, depending on whether the Update with Differences or Replace with Current Results option is selected for the inventory profile. Operational data tables reflect only the most current scan. However, if you enable history tracking, new, modified, and deleted data from the operational data tables are stored in history tables as the operational data tables are overwritten.

You can access historical data by using history views, queries, and tables. History views, queries, and tables are designated with an uppercase H and an underscore character (H_) at the beginning of their names. Following the H_, the name of a history view, query, or table generally has the same name as the operational view, query, or table with which it is associated.

Each operational data table that contains the column COMPUTER_SYS_ID has a corresponding history table. A history table has all of the column names from the corresponding operational data table plus two additional columns, RECORD_ACTION and PRFL_ACTION. The RECORD_ACTION column tells whether the record is an INSERT (new information is being added to the operational data table), an UPDATE (part of a record in the operational data table is being modified), or a DELETE (the record no longer exists in the operational data table). PRFL_ACTION states whether the profile configuration option is Replace with Current Results (REPLACE) or Update With Differences (REPLACE_WITH_DIFF). Also, while the column RECORD_TIME in operational data tables contains the time that the record was inserted into the database, RECORD_TIME in history tables contains the time that the endpoint was scanned (the COMPUTER_SCANTIME column from the COMPUTER table).

To enable history tracking, you must create history tables in the configuration repository. See Chapter 3, “Creating the Configuration Repository” for more information about creating these tables.

Note: Data is never deleted from the history tables. Even if you remove a system from the TMR using the **winvrmnode** command, the record of the system's existence remains in the history tables unless you manually delete the history tables. See "Deleting History Tables" on page C-4 for more information.

Using History Tables

When you enable history tracking for the first time, a series of history tables is installed in the configuration repository. After the history tables are installed, you should populate the history tables with information from a full scan. If you install the historical schema at the same time as the operational data schema, use the Update with Differences profile option for all scans. If you install the historical schema after the operational data schema, use the Replace with Current Results profile option for the first scan, then use Update with Differences for all successive scans. This ensures that only the change data is sent to the history tables. If you use Replace with Current Results for each scan, the size of the history tables will grow by the size of the full scan for each scan, and it will be more difficult to query the actual changes to the TMR.

In the following example, the operational data table MATCHED_SWARE stores data about installed software packages that have been found using signature matching at the endpoint. On the initial scan, the MATCHED_SWARE table returns 100 entries for a particular system, so the H_MATCHED_SWARE table also has 100 entries. Prior to the next scan, five software packages are added and five are deleted from that system.

If you use the Update with Differences option on the next scan, H_MATCHED_SWARE only adds 10 entries, five as INSERT entries for the five new software packages and five as DELETE entries for the five packages that were removed. However, if you use the Replace with Current Results option, the H_MATCHED_SWARE table adds 100 entries. Combined with the original 100 entries, the Replace with Current Results option leaves you with 200 total entries after the second scan, all of which are stored as INSERT entries. None of the entries explicitly show that some software packages were deleted. On the other hand, the Update with Differences option leaves you with only 110 total

entries, and the additional entries show which software packages were added and which were deleted.

Custom History Tables

Tivoli Inventory automatically populates custom history tables when the tables are properly added to the configuration repository. See Chapter 7, "Collecting Custom Information with Tivoli Inventory" for information about creating custom history tables.

Deleting History Tables

You can manually delete history tables either by time or by specific computer ID. The following examples show how to delete history tables for DB2.

To delete history tables for a specific system, enter the following:

```
delete from H_TABLE_NAME where COMPUTER_SYS_ID = 'SYSTEM_ID';
```

where:

H_TABLE_NAME
: Specifies the name of the history table.

SYSTEM_ID
: Specifies the computer system ID.

All history tables with the name specified by *H_TABLE_NAME* for the system with this computer system ID will be deleted when this command is run. To delete other history tables for the system with this computer system ID, you must run this command for each table name you want to delete.

To delete history tables before a specific time, enter the following:

```
delete from H_TABLE_NAME where RECORD_TIME < 'YYYY-MM-DD HH:MM:SS'
```

where:

H_TABLE_NAME
: Specifies the name of the history table.

YYYY-MM-DD HH:MM:SS
: Specifies the date and time before which you want to delete history tables. This format is for DB2; other platforms may require different formats.

All history tables with the name specified by *H_TABLE_NAME* and created before this date will be deleted when this command is run.

Configuration Repository Views

The following sections describe the pre-defined views provided with Tivoli Inventory. These views were created during the Tivoli Inventory installation. Each section describes the columns in each view and shows the platforms on which the data is collected. A check mark (✓) indicates the column *may* be populated when a view is run on that platform.

Note: A check mark does not indicate that a column will be populated every time, only that it is possible for data to be collected on that platform. A column may still remain unpopulated in some circumstances.

If no check mark is provided, the column *never* contains data for that platform. Exceptions are indicated with other symbols and explained in each section.

Note: Tivoli Inventory, Version 4.0, uses the same naming conventions for all databases. All table and column names are 18 characters or less in length and follow the abbreviation scheme used by RIM (see "Naming Custom Tables, Views, and Columns for DB2 and Informix" on page 7-16).

CDROM_VIEW

Displays information about CD-ROM drives on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓			✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓			✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓			✓	✓	✓
MANUFACTURER	The manufacturer of the installed CD-ROM drive.	✓					✓	✓
MODEL	The model of the installed CD-ROM drive.	✓	✓			✓	✓	✓
STORAGE_TYPE	The type of CD-ROM drive installed.	✓	✓			✓	✓	✓
SER_NUM	The serial number of the installed CD-ROM drive.	✓						
RECORD_TIME	The time the data was inserted into the database.	✓	✓			✓	✓	✓

COMPUTER_MEM_VIEW

Displays information about the installed memory on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
PHYSICAL_TOTAL_KB	The total amount of installed memory in KB.	✓	✓	✓	✓	✓	✓	✓
PHYSICAL_FREE_KB	The amount of free memory on the system in KB.		✓			✓	✓	✓
TOTAL_PAGES	The number of memory pages available on the system.	✓	✓		✓	✓	✓	✓
FREE_PAGES	The number of free memory pages on the system.	✓	✓			✓	✓	✓
PAGE_SIZE	The size of the page in bytes.	✓	✓			✓	✓	✓
VIRT_TOTAL_KB	The total amount of virtual memory on the system in KB.	✓	✓		✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
VIRT_FREE_KB	The amount of free virtual memory in the system in KB.	✓	✓			✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

COMPUTER_VIEW

Displays information about computer type and operating system information. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_MODEL	The system's model information.	✓	✓			✓	✓	✓
OS_NAME	The operating system (such as Windows or Solaris) that is running at the time of the scan.	✓	✓	✓	✓	✓	✓	✓
OS_TYPE	The type of operating system (such as Windows NT or Windows 2000) that is running at the time of the scan.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SCANTIME	The time of the last Tivoli Inventory scan.	✓	✓	✓	✓	✓	✓	✓

FLPY_DRV_VIEW

Displays information about floppy drives on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓			✓		✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓			✓		✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓			✓		✓	✓
MANUFACTURER	The manufacturer of the installed floppy drive.						W	W
MODEL	The model of the installed floppy drive.						W	W
STORAGE_TYPE	The type of floppy drive installed.	✓			✓		✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓			✓		✓	✓

[W] Reported only on systems *with* WMI.

HDISK_VIEW

Displays information about the hard drives installed in the system. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓		✓	✓	✓	✓
MANUFACTURER	The manufacturer of the installed hard drive.	✓					W	
MODEL	The model of the installed hard drive.	✓	✓			✓	W	✓
STORAGE_TYPE	The type of hard drive installed.	✓	✓		✓	✓	✓	✓
SER_NUM	The serial number of the installed hard drive.	✓						
HDISK_CYLINDERS	The number of cylinders in the hard drive.	*	*		✓	✓	✓	✓
HDISK_SECTORS	The number of sectors in the hard drive.	*	*		✓	✓	✓	✓

[W] Reported only on systems *with* WMI.
*** For AIX systems, this value is reported as 0. For HP systems, this value is reported as -1.**

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
HDISK_HEADS	The number of disk heads in the hard drive.	*	*		✓	✓	✓	✓
HDISK_SIZE_MB	The size in megabytes (MB) of the hard drive.	✓	✓		✓	✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓		✓	✓	✓	✓

[W] Reported only on systems *with* WMI.
*** For AIX systems, this value is reported as 0. For HP systems, this value is reported as -1.**

HEADER_INFO_VIEW

Displays header information about files installed on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.						✓	✓
TME_OBJECT_ID	The TME object ID for the system.						✓	✓
COMPUTER_SYS_ID	The ID of the system.						✓	✓
HEADER_NAME	The name of the software.						✓	✓
HEADER_VERS	The version number of the software.						✓	✓
HEADER_PUBLISHER	The publisher of the software.						✓	✓
RECORD_TIME	The time the data was inserted into the database.						✓	✓

INST_FILE_VIEW

Displays information about files installed on target systems. The columns in this view are as follows:

Note: On NetWare, OS/2, and Windows 98 systems, the CREATED_TIME, MODIFIED_TIME, and ACCESSED_TIME columns can all be populated depending upon how many of the time values the system supports.

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
FILE_NAME	The name of the file.	✓	✓	✓	✓	✓	✓	✓
FILE_SIZE	The size of the file.	✓	✓	✓	✓	✓	✓	✓
PATH	The path for the file.	✓	✓	✓	✓	✓	✓	✓
CREATED_TIME	The time the file was created.	✓	✓	✓	✓	✓	✓	✓
MODIFIED_TIME	The time the file was last modified.	✓	✓	✓	✓	✓	✓	✓
ACCESSED_TIME	The time the file was last accessed.	✓	✓	✓	✓	✓	✓	✓
FILE_PERMISSIONS	The permissions for the file.	✓	✓	✓	✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
FILE_OWNER	The owner of the file.	✓	✓			✓		
FILE_GROUP	The file group that contains the file.	✓	✓			✓		
CHECKSUM_QUICK	The 32-bit Quick CRC checksum value.	✓	✓	✓	✓	✓	✓	✓
CHECKSUM_CRC32	The 32-bit Full CRC checksum value.	✓	✓	✓	✓	✓	✓	✓
CHECKSUM_MD5	The 128-bit checksum value.	✓	✓	✓	✓	✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

INST_SWARE_VIEW

Displays information about software components installed on target systems that are matched at the endpoint using a software signature scan. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
SWARE_DESC	The software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_VERS	The version of the software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_NAME	The name of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_SIZE	The size of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

INVENTORYDATA

Displays information about the general hardware, memory, and operating system of target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_MODEL	The system's model information.	✓	✓			✓	✓	✓
PHYSICAL_TOTAL_KB	The total amount of installed memory in KB.	✓	✓	✓	✓	✓	✓	✓
PHYSICAL_FREE_KB	The amount of free memory on the system in KB.		✓			✓	✓	✓
TOTAL_PAGES	The number of memory pages available on the system.	✓	✓		✓	✓	✓	✓
FREE_PAGES	The number of free memory pages on the system.	✓	✓			✓	✓	✓
PAGE_SIZE	The size of the page in bytes.	✓	✓			✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
VIRT_TOTAL_KB	The total amount of virtual memory on the system.	✓	✓		✓	✓	✓	✓
VIRT_FREE_KB	The amount of free virtual memory in the system.	✓	✓			✓	✓	✓
PROCESSOR_MODEL	The model for the processor.	✓	✓	✓	✓	✓	✓	✓
PROCESSOR_SPEED	The current speed at which this processor is running.	✓	✓	✓	✓	✓	✓	✓
OS_NAME	The operating system (such as Windows or Solaris) that is running at the time of the scan.	✓	✓	✓	✓	✓	✓	✓
OS_TYPE	The type of operating system (such as Windows NT or Windows 2000) that is running at the time of the scan.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SCANTIME	The time of the last Tivoli Inventory scan.	✓	✓	✓	✓	✓	✓	✓

IP_ADDR_VIEW

Displays information about the internet protocol (IP) address on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓		✓	✓	✓	✓
IP_ADDR	The IP address for the system.	✓	✓		✓	✓	✓	✓
IP_HOSTNAME	The IP host name for the system.	✓	✓			✓	✓	✓
IP_DOMAIN	The IP domain name for the system.	✓	✓			✓	✓	✓
IP_SUBNET	The IP subnet for the system.	✓	✓			✓	✓	✓
IP_GATEWAY	The IP gateway name for the system.	✓	✓			✓	✓	✓
IP_PRIMARY_DNS	The primary domain name service (DNS) for the system.	✓	✓			✓	✓	✓
IP_SECONDARY_DNS	The secondary DNS for the system.	✓	✓			✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
RECORD_TIME	The time the data was inserted into the database.	✓	✓		✓	✓	✓	✓

IPX_ADDR_VIEW

Displays information about the IPX address on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.			✓			✓	✓
TME_OBJECT_ID	The TME object ID for the system.			✓			✓	✓
COMPUTER_SYS_ID	The ID of the system.			✓			✓	✓
IPX_ADDR	The IPX address of the system.			✓			✓	✓
NET_NUM	The net number of the system.						✓	✓
NODE_ADDR	The node address of the system.						✓	✓
LINK_SPEED	The link speed of the system.						*	*
MAX_PACKET_SIZE	The maximum packet size the system can handle.						*	*
RECORD_TIME	The time the data was inserted into the database.			✓			✓	✓

*** Not reported for systems running the Windows Management Interface (WMI).**

KEYBOARD_VIEW

Displays information about keyboards on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.				✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.				✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.				✓	✓	✓	✓
KEYBOARD_TYPE	The type of keyboard attached to the system.				✓	✓	✓	✓
FUNCTION_KEYS	The number of function keys on the keyboard.						✓	✓

MATCH_SWARE_VIEW

Displays information about installed software components by comparing unmatched file information with the SWARE_SIG table. While this view provides the same information as INST_SWARE_VIEW when scanning for installed software using signature matching, MATCH_SWARE_VIEW uses an older matching method. Tivoli recommends using INST_SWARE_VIEW. The columns in MATCH_SWARE_VIEW are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
SWARE_DESC	The description of the software package.	✓	✓	✓	✓	✓	✓	✓
SWARE_VERS	The version of the software package.	✓	✓	✓	✓	✓	✓	✓
SWARE_NAME	The name of the file associated with this software package.	✓	✓	✓	✓	✓	✓	✓
SWARE_SIZE	The size of the file associated with this software package.	✓	✓	✓	✓	✓	✓	✓

MOUSE_VIEW

Displays information about mouse pointer devices on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓			✓		✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓			✓		✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓			✓		✓	✓
BUTTONS	The number of buttons on the mouse.	✓			✓		✓	✓
MOUSE_MODEL	The model for the mouse.						✓	✓
MOUSE_TYPE	The class of pointing device (mouse, trackball, and so on) for the mouse.	✓			✓		✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓			✓		✓	✓

NATIV_SWARE_VIEW

Displays operating system product information on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓			✓	✓	✓
PACKAGE_NAME	The name of this software package.	✓	✓			✓	✓	✓
PACKAGE_VERS	The version for this software package.	✓	✓			✓	✓	✓
PUBLISHER	The publisher of this software package.					✓	✓	✓
PACKAGE_ID	The package ID for this software package.	✓	✓			✓	✓	✓
FILE_PATH	The path of the native software installed.					✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓			✓	✓	✓

NET_CARD_VIEW

Displays information about the network cards on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓		✓	✓	✓	✓
PERM_MAC_ADDR	The permanent MAC address for the system.	✓	✓			✓	✓	✓
CURRENT_ADDR	The current network address for the system.	✓	✓			✓	✓	✓
ADAPTER_TYPE	The network adapter installed on the system.	✓	✓				✓	✓
ADAPTER_MODEL	The model of the network adapter installed on the system.	✓			✓	✓	✓	✓
MANUFACTURER	The manufacturer of the network adapter installed on the system.				✓	✓	✓	✓
INST_DATE	The date that the network card was installed on the system.					W	W	W

[W] Reported only on systems *with* WMI.

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
RECORD_TIME	The time the data was inserted into the database.	✓	✓		✓	✓	✓	✓
[W] Reported only on systems *with* WMI.								

NOSIG_FILES_VIEW

Displays information about installed software files that do not match any software signatures. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
FILE_NAME	The file name for a file that does not match a software signature.	✓	✓	✓	✓	✓	✓	✓
FILE_SIZE	The size of a file that does not match a software signature.	✓	✓	✓	✓	✓	✓	✓
PATH	The path for a file that does not match a software signature.	✓	✓	✓	✓	✓	✓	✓

NW_SERVER_VIEW

Displays information about operating systems of NetWare servers. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.			✓				
TME_OBJECT_ID	The TME object ID for the system.			✓				
COMPUTER_SYS_ID	The ID of the system.			✓				
NW_DEV_NAME	The NetWare device name on the system.			✓				
NW_VERS	The major version number of the NetWare software installed on the system.			✓				
NW_SUB_VERS	The minor version number of the NetWare software installed on the system.			✓				
NW_MAX_CONNS	The maximum number of connections allowed on the system.			✓				
NW_MAX_VOLS	The maximum number of volumes allowed on the system.			✓				

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
NW_REVISION_LEVEL	The revision level of the NetWare software installed on the system.			✓				
NW_SFT_LEVEL	The NetWare System Fault Tolerant (SFT) level installed on the system.			✓				
NW_TTS_LEVEL	The NetWare Transaction Tracking System (TTS) level installed on the system.			✓				
NW_MAX_CONNS_USED	The maximum number of connections used on the system.			✓				
NW_ACCOUNTING_ VERS	The NetWare accounting version installed on the system.			✓				
NW_VAP_VERS	The value-added process (VAP) version installed on the system.			✓				
NW_QUEING_VERS	The queuing version installed on the system.			✓				
NW_PRINTSERVR_VERS	The print server version installed on the system.			✓				
NW_VIRT_CONS	The virtual console installed on the system.			✓				

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
NW_SEC_LEVEL	The security restriction level on the system.			✓				
NW_INET_BRG_SUPP	The internet bridge support installed on the system.			✓				
NW_CLIB_MAJOR_VERS	The major version number of the C runtime library installed on the system.			✓				
NW_CLIB_MINOR_VERS	The minor version number of the C runtime library installed on the system.			✓				
NW_CLIB_REVISION	The revision level of the C runtime library installed on the system.			✓				
NW_SER_NUM	The serial number of the NetWare software installed on the system.			✓				
RECORD_TIME	The time the data was inserted into the database.			✓				

NW_VOLS_VIEW

Displays information about the NetWare volumes on NetWare servers. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.			✓				
TME_OBJECT_ID	The TME object ID for the system.			✓				
COMPUTER_SYS_ID	The ID of the system.			✓				
NWVOL_NAME	The name of the NetWare volume installed on the system.			✓				
NWVOL_TOTAL_BLKS	The total blocks on the NetWare volume installed on the system.			✓				
NWVOL_BLK_SECTORS	The number of sections per block on the NetWare volume installed on the system.			✓				
NWVOL_AVAIL_BLKS	The number of available blocks on the NetWare volume installed on the system.			✓				
NWVOL_DIR_SLOTS	The total number of directory table entries available on the system.			✓				

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
NWVOL_AVAIL_SLOTS	The number of available slots on the NetWare volume installed on the system.			✓				
NWVOL_IS_REMOVABLE	Whether the NetWare volume installed is removable.			✓				
RECORD_TIME	The time the data was inserted into the database.			✓				

OS_VIEW

Displays information about operating systems on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
OS_NAME	The operating system (such as Windows or Solaris).	✓	✓	✓	✓	✓	✓	✓
OS_TYPE	The type of operating system (such as Windows NT or Windows 2000).	✓	✓	✓	✓	✓	✓	✓
OS_MAJOR_VERS	The major version number of the operating system.	✓	✓	✓	✓	✓	✓	✓
OS_MINOR_VERS	The minor version number of the operating system.	✓	✓	✓	✓	✓	✓	✓
OS_SUB_VERS	The subversion number of the operating system.					✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
OS_INST_DATE	The date on which the operating system was installed.				✓		✓	✓

PARTITION_VIEW

Displays information about the disk partitions on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓		✓	✓	✓	✓
FS_ACCESS_POINT	The location where the partition is mounted.	✓	✓		✓	✓	✓	✓
DEV_NAME	The device name.	✓	✓		✓	✓	✓	✓
PARTITION_TYPE	The type of partition on the drive.	✓	✓		✓	✓	✓	✓
MEDIA_TYPE	The media type that contains the partition.	✓	✓		✓	✓	✓	✓
PHYSICAL_SIZE_KB	The size of the drive that contains the partition.	✓	✓		✓	✓	✓	✓
FS_TYPE	The file system type.	✓	✓		✓	✓	✓	✓
FS_MOUNT_POINT	The point where the partition attaches to the operating system.	✓	✓		✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
FS_TOTAL_SIZE_KB	The size of the partition in KB.	✓	✓		✓	✓	✓	✓
FS_FREE_SIZE_KB	The amount of free space on the partition in KB.	✓	✓		✓	✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓		✓	✓	✓	✓

PC_BIOS_VIEW

Displays information about BIOS of PCs. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.						✓	✓
TME_OBJECT_ID	The TME object ID for the system.						✓	✓
COMPUTER_SYS_ID	The ID of the system.						✓	✓
USER_NAME	The user name for the system.						✓	✓
DOMAIN_NAME	The domain name for the system.						✓	✓
WORKGROUP_NAME	The workgroup name for the system.						✓	W
BIOS_ID	The BIOS ID for the system.						✓	✓
BIOS_ID_BYTES	The size of the BIOS ID in bytes.						✓	✓
BIOS_DATE	The revision date of the BIOS.						✓	✓
BIOS_STRING	The description of the BIOS.						✓	✓

W Reported only on systems *with* WMI.

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
BIOS_MANUFACTURER	The BIOS manufacturer.						✓	✓
BIOS_MODEL	The BIOS model.						✓	✓
BIOS_SER_NUM	The BIOS serial number.						✓	✓
RECORD_TIME	The time the data was inserted into the database.						✓	✓

[W] Reported only on systems *with* WMI.

PC_PROCESSOR_VIEW

Displays information about processors on target PCs. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.			✓	✓		✓	✓
TME_OBJECT_ID	The TME object ID for the system.			✓	✓		✓	✓
COMPUTER_SYS_ID	The ID of the system.			✓	✓		✓	✓
MANUFACTURER	The manufacturer for the processor.			✓	✓		✓	✓
PROCESSOR_MODEL	The model for the processor.			✓	✓		✓	✓
PROCESSOR_SPEED	The current speed at which the processor is running.			✓	✓		✓	✓
CHIP_FAMILY	The chip family for the processor.			✓	✓		✓	✓
CHIP_MODEL	The chip model for the processor.			✓	✓		✓	✓
CHIP_STEPPING	The chip stepping setting for the processor.			✓	✓		✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
VIRT_MODE_EXT	The virtual mode extensions for the processor.			✓	✓		✓	✓
PAGE_SIZE_EXT	The page size extensions for the processor.			✓	✓		✓	✓
TIME_STAMP_COUNTER	The time stamp counter for the processor.			✓	✓		✓	✓
MODEL_SPECIFIC_REG	The model-specific registers for the processor.			✓	✓		✓	✓
PHYSICAL_ADDR_EXT	The physical address extensions for the processor.			✓	✓		✓	✓
MACHINECHECK_EXCPT	The machine check exceptions for the processor.			✓	✓		✓	✓
CMPXCHG8B_SUPP	The CMPXCHG8B instruction support for the processor.			✓	✓		✓	✓
ON_CHIP_APIC	The integrated advanced programmable interrupt controller (APIC) for the processor.			✓	✓		✓	✓
MEM_TYPE_RANGE_ REG	The memory type range registers for the processor.			✓	✓		✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
PAGE_GLOBAL_ENABLE	The page global enable setting for the processor.			✓	✓		✓	✓
MACHINECHECK_ARCH	The machine check architecture for the processor.			✓	✓		✓	✓
COND_MOVE_SUPP	The conditional move instruction setting for the processor.			✓	✓		✓	✓
MMX_TECHNOLOGY	The Intel MMX features for the processor (if any).			✓	✓		✓	✓
ON_CHIP_FPU	The integrated floating processor unit (FPU) for the processor.			✓	✓		✓	✓
DEBUG_EXT_PRESENT	Whether the processor has debug extensions.			✓	✓		✓	✓
FAST_SYS_CALL	The fast system call setting for the processor.			✓	✓		✓	✓
PAGE_ATTR_TABLE	The page attribute table for the processor.			✓	✓		✓	✓
PAGE_SIZE_EXT36	The 36-bit page size extension for the processor.			✓	✓		✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
SER_NUM_ENABLED	Whether the process serial number is enabled.			✓	✓		✓	✓
FAST_FLOAT_SAVE	The fast floating point save/restore setting for the processor.			✓	✓		✓	✓
SIMD_EXT_SUPP	The streaming single instruction/multiple data (SIMD) extensions support for the processor.			✓	✓		✓	✓
NOW_3_D_ARCH	The AMD 3DNow! features for the processor (if any).			✓	✓		✓	✓
SER_NUM	The serial number for the processor.			✓	✓		✓	✓
RECORD_TIME	The time the data was inserted into the database.			✓	✓		✓	✓

PCI_DEV_VIEW

Displays information about PCI devices installed in the system. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.						✓	✓
TME_OBJECT_ID	The TME object ID for the system.						✓	✓
COMPUTER_SYS_ID	The ID of the system.						✓	✓
INST_PCI_ID	The PCI ID of the PCI device installed in or connected to the system.						✓	✓
PCI_DEV_NAME	The name of the PCI device installed in or connected to the system.						✓	✓
RECORD_TIME	The time the data was inserted into the database.						✓	✓

PRINTER_VIEW

Displays information about printers attached to target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.						✓	✓
TME_OBJECT_ID	The TME object ID for the system.						✓	✓
COMPUTER_SYS_ID	The ID of the system.						✓	✓
PRINTER_MODEL	The model for the printer.						✓	✓
PRINTER_NAME	The name of the printer.						✓	✓
PRINTER_LOCATION	The location for the printer.						✓	✓
PRINTER_IS_LOCAL	Whether the printer is a local printer or a network printer.						✓	✓
DRV_NAME	The printer driver name for the printer.						✓	✓
DRV_VERS	The printer driver version for the printer.						✓	✓
PORT_NAME	The port name used by the printer.						✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
RECORD_TIME	The time the data was inserted into the database.						✓	✓

PROCESSOR_VIEW

Displays information about processors on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
MANUFACTURER	The manufacturer for the processor.		✓	✓	✓		✓	✓
PROCESSOR_MODEL	The model for the processor.	✓	✓	✓	✓	✓	✓	✓
PROCESSOR_SPEED	The current speed at which this processor is running.	✓	✓	✓	✓	✓	✓	✓
SER_NUM	The serial number for this processor.			✓	✓		✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

STORAGE_DEV_VIEW

Displays information about storage devices on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓		✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓		✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓		✓	✓	✓	✓
MANUFACTURER	The manufacturer of the installed storage device.	✓					✓	✓
MODEL	The model of the installed storage device.	✓	✓			✓	✓	✓
STORAGE_TYPE	The type of storage device installed.	✓	✓		✓	✓	✓	✓
SER_NUM	The serial number of the installed storage device.	✓						
RECORD_TIME	The time the data was inserted into the database.	✓	✓		✓	✓	✓	✓

SWARE_MATCH_CRC32

Displays information about files, based on their Full CRC32 cyclic redundancy check (CRC) value, that are installed on target systems. This view matches the CRC32 checksum value from the UNMATCHED_FILES table to the CRC32 checksum value from the SWARE_SIG table. You must run a scan for basic file information using with the Full option selected before running this view. See “winvsig” on page B-67 for information on setting the checksum value. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
SWARE_DESC	The software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_VERS	The version of the software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_NAME	The name of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
SWARE_SIZE	The size of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

SWARE_MATCH_MD5

Displays information about files, based on their MD5 checksum value, installed on target systems. This view matches the MD5 checksum value from the UNMATCHED_FILES table to the MD5 checksum value from the SWARE_SIG table. You must run a scan for basic file information using with the MD5 option selected before running this view. See "winvsig" on page B-67 for information on setting the checksum value. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
SWARE_DESC	The software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_VERS	The version of the software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_NAME	The name of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_SIZE	The size of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

SWARE_MATCH_QUICK

Displays information about files, based on their Quick checksum value, installed on target systems. This view matches the Quick checksum value from the UNMATCHED_FILES table to the Quick checksum value from the SWARE_SIG table. You must run a scan for basic file information using with the Quick option selected before running this view. See “winvsig” on page B-67 for information on setting the checksum value. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.	✓	✓	✓	✓	✓	✓	✓
TME_OBJECT_ID	The TME object ID for the system.	✓	✓	✓	✓	✓	✓	✓
COMPUTER_SYS_ID	The ID of the system.	✓	✓	✓	✓	✓	✓	✓
SWARE_DESC	The software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_VERS	The version of the software associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_NAME	The name of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓
SWARE_SIZE	The size of the file associated with this software signature.	✓	✓	✓	✓	✓	✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
RECORD_TIME	The time the data was inserted into the database.	✓	✓	✓	✓	✓	✓	✓

USB_DEV_VIEW

Displays information about USB devices. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.						✓	*
TME_OBJECT_ID	The TME object ID for the system.						✓	*
COMPUTER_SYS_ID	The ID of the system.						✓	*
HOST_CNTRL	The host controller for the USB device.						✓	*
DEV_ADDR	The device address for the USB device.						✓	*
SER_NUM	The serial number for the USB device.						✓	*
PORT_NUM	The port number used by the USB device.						✓	*
PARENT_ADDR	The parent address used by the USB device.						✓	*
USB_VERS	What kind of USB device this device is (for example, a USB 1.1 device or a USB 2.0 device).						✓	*

*** Windows NT does not support USB devices.**

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
DEV_CLASS	The device class for the USB device.						✓	*
DEV_SUBCLASS	The device subclass for the USB device.						✓	*
VENDOR_ID	The manufacturer's vendor ID for the USB device.						✓	*
PRODUCT_ID	The product ID for the USB device.						✓	*
MANUFACTURER	The manufacturer for the USB device.						✓	*
PRODUCT	The USB product.						✓	*
NUM_OF_PORTS	The number of USB ports present on the USB device.						✓	*
DEV_IS_HUB	Whether the USB device is a USB hub or not.						✓	*
RECORD_TIME	The time the data was inserted into the database.						✓	*

*** Windows NT does not support USB devices.**

VID_CARD_VIEW

Displays information about video cards on target systems. The columns in this view are as follows:

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
TME_OBJECT_LABEL	The TME object label for the system.				✓		✓	✓
TME_OBJECT_ID	The TME object ID for the system.				✓		✓	✓
COMPUTER_SYS_ID	The ID of the system.				✓		✓	✓
VID_CARD_MODEL	The manufacturer for the video card.				✓		✓	✓
VID_CARD_BIOS	The BIOS information for the video card.						✓	✓
VID_DAC_TYPE	The integrated digital-to-analog converter (DAC) for the video card.						✓	✓
VID_MEM	The amount of memory for the video card.				✓		✓	✓
VID_BIOS_RELDATE	The release date of the BIOS for the video card.						✓	✓
VID_CHIP_TYPE	The chip type for the video card.						✓	✓

Column Name	Description	AIX	HP	NetWare	OS/2	Solaris	Win 98	Win NT/2000
VID_HORIZNTL_RES	The horizontal resolution setting of the installed video card.						✓	✓
VID_VERTICAL_RES	The vertical resolution setting of the installed video card.						✓	✓
VID_COLORS	The color setting of the installed video card.						✓	✓
RECORD_TIME	The time the data was inserted into the database.				✓		✓	✓

Historical Configuration Repository Views

Generally, the names of historical configuration repository views add H_ to the beginning of the name of the non-historical view on which they are based. A historical view returns all of the columns returned by its non-historical view plus the following two columns:

Column Name	Description
RECORD_ACTION	Whether the record is an INSERT (new information is being added to the operational data table), an UPDATE (some part of a record in the operational data table is being modified), or a DELETE (the record no longer exists in the operational data table).

Column Name	Description
PRFL_ACTION	Whether the profile configuration option was REPLACE (Replace with Current Results) or REPLACE_WITH_DIFF (Update with Differences).

The following list shows the available historical views:

H_CDROM_VIEW
H_COMPUTER_VIEW
H_FLPY_DRV_VIEW
H_HDISK_VIEW
H_HEADER_VIEW
H_INST_FILE_VIEW
H_INST_SWARE_VIEW
H_IP_ADDR_VIEW
H_IPX_ADDR_VIEW
H_MEM_VIEW
H_MOUSE_VIEW
H_NATIV_VIEW
H_NET_CARD_VIEW
H_NOSIG_FILES_VIEW
H_NW_SERVER_VIEW
H_NW_VOLS_VIEW
H_OS_VIEW
H_PARTITION_VIEW
H_PC_BIOS_VIEW
H_PCI_DEV_VIEW
H_PCPROCESSOR_VIEW
H_PRINTER_VIEW
H_PROCESSOR_VIEW
H_STORAGE_DEV_VIEW

H_TAPEDRV_VIEW

H_UNIX_SYS_VIEW

H_USB_DEV_VIEW

H_VID_CARD_VIEW

See the associated non-historical view for a full description of each view and the columns returned with the view.

Tivoli Inventory and Subscription Queries

The following sections list the pre-defined queries provided with Tivoli Inventory and the columns included in each. See Chapter 2, “Installing Tivoli Inventory, Version 4.0” for instructions on creating the Tivoli Inventory queries. See Chapter 8, “Querying Inventory Information” for instructions on creating your own queries.

Tivoli Inventory Queries

The following list describes the pre-defined queries and shows the included columns provided with Tivoli Inventory. For descriptions of the column names, refer to the view against which the query is run.

CDROM_QUERY

This query returns information on the model and physical details of CD-ROM drives installed on systems. It runs against the view CDROM_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

MODEL

STORAGE_TYPE

SER_NUM

RECORD_TIME

COMPUTER_MEM_QUERY

This query returns information about the memory installed on a system. It runs against the view COMPUTER_MEM_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

PHYSICAL_TOTAL_KB

PHYSICAL_FREE_KB

TOTAL_PAGES

FREE_PAGES

PAGE_SIZE

VIRT_TOTAL_KB

VIRT_FREE_KB

RECORD_TIME

FLPY_DRV_QUERY

This query returns information on the model and physical details of floppy disk drives installed on systems. It runs against the view FLPY_DRV_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

MODEL

STORAGE_TYPE

RECORD_TIME

HDISK_QUERY

This query returns information on the model and physical details of hard disks installed on systems. It runs against the view HDISK_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

MODEL

STORAGE_TYPE

SER_NUM

HDISK_CYLINDERS

HDISK_SECTORS

HDISK_HEADS

HDISK_SIZE_MB

RECORD_TIME

HEADER_INFO_QUERY

This query returns basic software header information. It runs against the view HEADER_INFO_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

HEADER_NAME

HEADER_VERS

HEADER_PUBLISHER

RECORD_TIME

INST_FILE_QUERY

Note: This query is very large and requires a long time to run.

This query returns basic information about installed files on a system. It runs against the view INST_FILE_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

FILE_NAME

FILE_SIZE

PATH

CREATED_TIME

MODIFIED_TIME

ACCESSED_TIME

FILE_PERMISSIONS

FILE_OWNER

FILE_GROUP

CHECKSUM_QUICK

CHECKSUM_CRC32

CHECKSUM_MD5

RECORD_TIME

INVENTORY_HWARE

This query returns basic inventory hardware information for systems. It runs against the view INVENTORYDATA and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

COMPUTER_SCANTIME

COMPUTER_MODEL

OS_NAME

OS_TYPE

PROCESSOR_MODEL

PROCESSOR_SPEED

PHYSICAL_TOTAL_KB

PHYSICAL_FREE_KB

TOTAL_PAGES

FREE_PAGES

PAGE_SIZE

VIRT_TOTAL_KB

VIRT_FREE_KB

INVENTORY_SWARE

This query returns basic software information. It runs against the view INST_SWARE_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

SWARE_DESC

SWARE_VERS

SWARE_NAME

SWARE_SIZE

RECORD_TIME

IP_ADDR_QUERY

This query returns details on the IP address for a system. It runs against the view IP_ADDR_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

IP_ADDR

IP_HOSTNAME

IP_DOMAIN

IP_SUBNET

IP_GATEWAY

IP_PRIMARY_DNS

IP_SECONDARY_DNS

RECORD_TIME

IPX_ADDR_QUERY

This query returns details on the IPX address for a system. It runs against the view IPX_ADDR_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

IPX_ADDR

NET_NUM

NODE_ADDR

LINK_SPEED

MAX_PACKET_SIZE

RECORD_TIME

KEYBOARD_QUERY

This query returns information about the keyboard installed on a system. It runs against the view KEYBOARD_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

KEYBOARD_TYPE

FUNCTION_KEYS

MATCH_CRC32_QUERY

Note: This query is very large and requires a long time to run.

This query returns basic information on software that matches a software signature based on a CRC32 checksum match. It runs against the view SWARE_MATCH_CRC32 and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

SWARE_DESC

SWARE_VERS

SWARE_NAME

SWARE_SIZE

CHECKSUM_CRC32

RECORD_TIME

MATCH_MD5_QUERY

Note: This query is very large and requires a long time to run.

This query returns basic information on software that matches a software signature based on a MD5 checksum match. It runs against the view SWARE_MATCH_MD5 and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

SWARE_DESC

SWARE_VERS

SWARE_NAME

SWARE_SIZE

CHECKSUM_MD5

RECORD_TIME

MATCH_QUICK_QUERY

Note: This query is very large and requires a long time to run.

This query returns basic information on software that matches a software signature based on a quick checksum match. It runs against the view SWARE_MATCH_QUICK and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

SWARE_DESC

SWARE_VERS

SWARE_NAME

SWARE_SIZE

CHECKSUM_QUICK

RECORD_TIME

MOUSE_QUERY

This query returns information on the mouse installed on a system. It runs against the view MOUSE_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

BUTTONS

MOUSE_MODEL

MOUSE_TYPE

RECORD_TIME

NATIV_SWARE_QUERY

This query returns basic software information. It runs against the view NATIV_SWARE_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

PACKAGE_NAME

PACKAGE_VERS

PUBLISHER

PACKAGE_ID

RECORD_TIME

NET_CARD_QUERY

This query returns information on network cards installed on systems. It runs against the view NET_CARD_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

PERM_MAC_ADDR

CURRENT_ADDR

ADAPTER_TYPE

ADAPTER_MODEL

MANUFACTURER

INST_DATE

NOSIG_FILES_QUERY

Note: This query is very large and requires a long time to run.

This query returns basic file information for files that have no matching software signature. It runs against the view NOSIG_FILES_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

FILE_NAME

FILE_SIZE

NW_SERVER_QUERY

This query returns details of the operating system installed on NetWare servers. It runs against the view

NW_SERVER_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

NW_DEV_NAME

NW_VERS

NW_SUB_VERS

NW_MAX_CONNS

NW_MAX_VOLS

NW_REVISION_LEVEL

NW_SFT_LEVEL

NW_TTS_LEVEL

NW_MAX_CONNS_USED

NW_ACCOUNTING_VERS

NW_VAP_VERS

NW_QUEING_VERS

NW_PRINTSERVR_VERS

NW_VIRT_CONS

NW_SEC_LEVEL

NW_INET_BRG_SUPP

NW_CLIB_MAJOR_VERS

NW_CLIB_MINOR_VERS

NW_CLIB_REVISION

NW_SER_NUM

RECORD_TIME

NW_VOLS_QUERY

This query returns information on NetWare volumes for NetWare systems. It runs against the view NW_VOLS_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

NWVOL_NAME

NWVOL_TOTAL_BLKS

NWVOL_BLK_SECTORS

NWVOL_AVAIL_BLKS

NWVOL_DIR_SLOTS

NWVOL_AVAIL_SLOTS

NWVOL_IS_REMOVABLE

RECORD_TIME

OS_QUERY

This query returns the information for an operating system. It runs against the view OS_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

OS_NAME

OS_TYPE

OS_MAJOR_VERS

OS_MINOR_VERS

OS_SUB_VERS

OS_INST_DATE

PARTITION_QUERY

This query returns information on the hard drive partitions on systems. It runs against the view PARTITION_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

FS_ACCESS_POINT

DEV_NAME

PARTITION_TYPE

MEDIA_TYPE

PHYSICAL_SIZE_KB

FS_TYPE

FS_MOUNT_POINT

FS_TOTAL_SIZE_KB

FS_FREE_SIZE_KB

RECORD_TIME

PC_BIOS_QUERY

This query returns the available BIOS information for a PC system. It runs against the view PC_BIOS_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

USER_NAME

DOMAIN_NAME

WORKGROUP_NAME

BIOS_ID

BIOS_ID_BYTES

BIOS_DATE

BIOS_STRING

BIOS_MANUFACTURER

BIOS_MODEL

BIOS_SER_NUM

RECORD_TIME

PC_PROCESSOR_QUERY

This query returns information about the installed processor model on a PC system. It runs against the view PC_PROCESSOR_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

PROCESSOR_MODEL

PROCESSOR_SPEED

SER_NUM

CHIP_FAMILY

CHIP_MODEL

CHIP_STEPPING

VIRT_MODE_EXT

PAGE_SIZE_EXT

TIME_STAMP_COUNTER

MODEL_SPECIFIC_REG

PHYSICAL_ADDR_EXT

MACHINECHECK_EXCPT

CMPXCHG8B_SUPP

ON_CHIP_APIC

MEM_TYPE_RANGE_REG

PAGE_GLOBAL_ENABLE

MACHINECHECK_ARCH

COND_MOVE_SUPP

MMX_TECHNOLOGY

ON_CHIP_FPU

DEBUG_EXT_PRESENT

FAST_SYS_CALL

PAGE_ATTR_TABLE

PAGE_SIZE_EXT36

SER_NUM_ENABLED

FAST_FLOAT_SAVE

SIMD_EXT_SUPP

NOW_3_D_ARCH

RECORD_TIME

PCI_DEV_QUERY

This query returns details on the PCI devices installed in a system. It runs against the view PCI_DEV_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

INST_PCI_ID

PCI_DEV_NAME

RECORD_TIME

PRINTER_QUERY

This query returns information on the printer attached to a system. It runs against the view PRINTER_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

PRINTER_MODEL

PRINTER_NAME

PRINTER_LOCATION

PRINTER_IS_LOCAL

DRV_NAME

DRV_VERS

PORT_NAME

RECORD_TIME

PROCESSOR_QUERY

This query returns information about the installed processor model on a system. It runs against the view PROCESSOR_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

PROCESSOR_MODEL

PROCESSOR_SPEED

SER_NUM

RECORD_TIME

STORAGE_DEV_QUERY

This query returns storage information for a system. It runs against the view STORAGE_DEV_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

MODEL

STORAGE_TYPE

SER_NUM

RECORD_TIME

TAPEDRV_QUERY

This query returns information on the model and physical details of tape drives installed on systems. It runs against the view TAPEDRV_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

MANUFACTURER

MODEL

STORAGE_TYPE

SER_NUM

RECORD_TIME

UNIX_SYS_QUERY

This query returns the available system information for a UNIX system. It runs against the view UNIX_SYS_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

BOOT_TIME

UPTIME

RUN_LEVEL

HOST_NAME

RECORD_TIME

USB_DEV_QUERY

This query returns details on the USB devices attached to a system. It runs against the view USB_DEV_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

HOST_CNTRL

DEV_ADDR

SER_NUM

PORT_NUM

PARENT_ADDR

USB_VERS

DEV_CLASS

DEV_SUBCLASS

VENDOR_ID

PRODUCT_ID

MANUFACTURER

PRODUCT

NUM_OF_PORTS

DEV_IS_HUB

RECORD_TIME

VID_CARD_QUERY

This query returns details on the video card installed on a system. It runs against the view VID_CARD_VIEW and returns the following columns:

TME_OBJECT_LABEL

TME_OBJECT_ID

COMPUTER_SYS_ID

VID_CARD_MODEL

VID_CARD_BIOS

VID_DAC_TYPE

VID_MEM

VID_BIOS_RELDATE

VID_CHIP_TYPE

VID_HORIZNTL_RES

VID_VERTICAL_RES

VID_COLORS

RECORD_TIME

Tivoli Historical Inventory Queries

The names of historical inventory queries add H_ to the name of the non-historical query on which they are based. A historical query returns all of the columns returned by its non-historical query plus the RECORD_ACTION and PRFL_ACTION columns.

The following list shows the available historical queries:

H_CDROM_QUERY

H_COMPUTER_MEM_QUERY

H_COMPUTER_QUERY

H_FLPY_DRV_QUERY

H_HDISK_QUERY

H_HEADER_INFO_QUERY

H_INST_FILE_QUERY

Note: H_INST_FILE_QUERY is very large and requires a long time to run.

H_INVENTORY_SWARE

H_IP_ADDR_QUERY

H_IPX_ADDR_QUERY

H_MOUSE_QUERY

H_NATIV_SWARE_QUERY

H_NET_CARD_QUERY

H_NOSIG_FILES_QUERY

Note: H_NOSIG_FILES_QUERY is very large and requires a long time to run.

H_NW_SERVER_QUERY

H_NW_VOLS_QUERY

H_OS_QUERY

H_PARTITION_QUERY

H_PC_BIOS_QUERY

H_PCI_DEV_QUERY

H_PCPROCESSOR_QUERY

H_PRINTER_QUERY

H_PROCESSOR_QUERY

H_STORAGE_DEV_QUERY

H_TAPEDRV_QUERY

H_UNIX_SYS_QUERY

H_USB_DEV_QUERY
H_VID_CARD_QUERY

Subscription Queries

The following is a list of the pre-defined subscription queries provided with Tivoli Inventory. See Chapter 8, “Querying Inventory Information,” for more information about subscription queries and creating your own queries.

The following queries return a list of systems that have the specified operating system:

Subscription Query	Description	View Run Against
SUB_128MB_SUBSCRIPTION	Query for systems that have 128 MB of memory or less installed.	INVENTORYDATA
128MB_SUBSCRIPTION	Query for systems that have more than 128 MB of memory installed.	INVENTORYDATA
256MB_SUBSCRIPTION	Query for systems that have more than 256 MB of memory installed.	INVENTORYDATA
512MB_SUBSCRIPTION	Query for systems that have more than 512 MB of memory installed.	INVENTORYDATA
AIX_SUBSCRIPTION	Query for all AIX systems.	COMPUTER_VIEW
HPUX_SUBSCRIPTION	Query for all HP-UX systems.	COMPUTER_VIEW
NETWARE_SUBSCRIPTION	Query for all NetWare systems.	COMPUTER_VIEW
OS2_SUBSCRIPTION	Query for all OS/2 systems.	COMPUTER_VIEW

Subscription Query	Description	View Run Against
Solaris_SUBSCRIPTION	Query for all Solaris systems.	COMPUTER_VIEW
WIN_2000_SUBSCRIPTION	Query for all Windows 2000 systems.	COMPUTER_VIEW
WIN_98_SUBSCRIPTION	Query for all Windows 98 systems.	COMPUTER_VIEW
WIN_ALL_SUBSCRIPTION	Query for all Windows systems.	COMPUTER_VIEW
WIN_ME_SUBSCRIPTION	Query for all Windows Me systems.	COMPUTER_VIEW
WIN_NT_SUBSCRIPTION	Query for all Windows NT systems.	COMPUTER_VIEW

All of these queries return the following columns:

TME_OBJECT_ID

TME_OBJECT_LABEL

Historical Subscription Queries

The following is a list of the pre-defined historical subscription queries provided with Tivoli Inventory. See Chapter 8, “Querying Inventory Information,” for more information about subscription queries and creating your own queries.

The following historical subscription queries return a list of systems that have the specified operating system:

Subscription Query	Description	View Run Against
H_AIX_SUBSCRIPTION	Historical query for all AIX systems.	COMPUTER_VIEW
H_HPUX_SUBSCRIPTION	Historical query for all HP-UX systems.	COMPUTER_VIEW
H_NETWARE_SUBSCRIPTION	Historical query for all NetWare systems.	COMPUTER_VIEW
H_OS2_SUBSCRIPTION	Historical query for all HP-UX systems.	COMPUTER_VIEW
H_Solaris_SUBSCRIPTION	Historical query for all Solaris systems.	COMPUTER_VIEW
H_WIN_2000_SUBSCRIPTION	Historical query for all Windows 2000 systems.	COMPUTER_VIEW
H_WIN_98_SUBSCRIPTION	Historical query for all Windows 98 systems.	COMPUTER_VIEW
H_WIN_ALL_SUBSCRIPTION	Historical query for all Windows systems.	COMPUTER_VIEW
H_WIN_ME_SUBSCRIPTION	Historical query for all Windows Me systems.	COMPUTER_VIEW
H_WIN_NT_SUBSCRIPTION	Historical query for all Windows NT systems.	COMPUTER_VIEW

All of the historical subscription queries return the following columns:

TME_OBJECT_ID

TME_OBJECT_LABEL

Configuration Repository Tables

This section describes the operational data tables populated by Tivoli Inventory in the configuration repository and lists their associated columns.

Note: Columns are described in "Configuration Repository Views" on page C-5. To locate a description for a column, refer to the column name in the index of this book.

Some of the tables in the configuration repository are not populated by Tivoli Inventory. Do not populate these tables with custom information because they are reserved for future enhancements. See Chapter 7, "Collecting Custom Information with Tivoli Inventory," for instructions on adding tables to the configuration repository.

COMPUTER

Definition

Stores common information about a computer system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

COMPUTER_SCANTIME

TME_OBJECT_ID

TME_OBJECT_LABEL

COMPUTER_MODEL

COMPUTER_BOOT_TIME

COMPUTER_ALIAS

SYS_SER_NUM

OS_NAME

OS_TYPE

OS_MAJOR_VERS

OS_MINOR_VERS

OS_SUB_VERS

OS_INST_DATE

REGISTERED_OWNER

REGISTERED_ORG

KEYBOARD_TYPE

FUNCTION_KEYS

TZ_LOCALE

TZ_NAME

TZ_DAYLIGHT_NAME

ON_SAVINGS_TIME

TZ_SECONDS

TIME_DIRECTION

RECORD_TIME

COMPUTER_SYS_MEM

Definition

Describes the physical and virtual memory installed on a system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

PHYSICAL_TOTAL_KB

PHYSICAL_FREE_KB

TOTAL_PAGES

FREE_PAGES

PAGE_SIZE

VIRT_TOTAL_KB

VIRT_FREE_KB

RECORD_TIME

FILE_DESC

Definition

Stores the file name and size for a file found from a basic file information scan; works with the UNMATCHED_FILES table. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory software scan.

Columns

FILE_DESC_ID

FILE_NAME

FILE_SIZE

FILE_FILTER

Definition

Contains the file name of a file filter. Software scanners use this table for filtering.

Comment

Populated by the **winvfilter** command and the GUI for Tivoli Inventory, Version 4.0.

Columns

FILE_NAME

FILE_PATH

Definition

Stores the path for a file found from a basic file information scan; works with the UNMATCHED_FILES table. A basic file information scan populates this table. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory software scan.

Columns

FILE_PATH_ID

PATH

HDISK

Definition

Stores the details for one particular type and model of hard drive; works with the STORAGE_DEV table. The configuration repository contains only one table for a specific device, even if the exact same device installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

HDISK_ID

HDISK_CYLINDERS

HDISK_SECTORS

HDISK_HEADS

HDISK_SIZE_MB

HEADER_INFO

Definition

Stores the details for a software header; works with the INST_HEADER_INFO table. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory software header scan (Windows).

Columns

HEADER_ID

HEADER_NAME

HEADER_VERS

HEADER_PUBLISHER

INST_HEADER_INFO

Definition

Contains the location of the HEADER_INFO table that contains details about the file. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory software header scan (Windows).

Columns

COMPUTER_SYS_ID

HEADER_ID

RECORD_TIME

INST_MOUSE

Definition

Describes the type of mouse installed, the operating system settings, and the location of the MOUSE table that contains details about the mouse. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

MOUSE_ID

INST_MOUSE_ID

RECORD_TIME

INST_NATIV_SWARE

Definition

Contains the file path and the location of the NATIV_SWARE table that contains details about the native software installed. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory software scan.

Columns

COMPUTER_SYS_ID

NATIV_ID

FILE_PATH

RECORD_TIME

INST_PARTITION

Definition

Describes a disk partition on a drive on the system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

FS_ACCESS_POINT

DEV_NAME

PARTITION_TYPE

MEDIA_TYPE

PHYSICAL_SIZE_KB

FS_TYPE

FS_MOUNT_POINT

FS_TOTAL_SIZE_KB

FS_FREE_SIZE_KB

RECORD_TIME

INST_PRINTER

Definition

Describes the type of printer installed, the driver software, and port settings. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

PRINTER_ID

INST_PRINTER_ID

PRINTER_NAME

PRINTER_LOCATION

PRINTER_IS_LOCAL

DRV_NAME

DRV_VERS

PORT_NAME

RECORD_TIME

INST_PROCESSOR

Definition

Describes the type of processor or processors installed. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

PROCESSOR_NUM

PROCESSOR_ID

SER_NUM

RECORD_TIME

INST_USB_DEV

Definition

Describes the type of USB device installed. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

USB_ID

HOST_CNTRL

DEV_ADDR

SER_NUM

PORT_NUM

PARENT_ADDR

RECORD_TIME

INST_VID_CARD

Definition

Describes the type of video card installed, and the operating system settings. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

VID_CARD_ID

INST_VID_CARD_ID

VID_HORIZNTL_RES

VID_VERTICAL_RES

VID_COLORS

RECORD_TIME

IP_ADDR

Definition

Describes the system's internet protocol (IP) address. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

IP_ADDR

IP_HOSTNAME

IP_DOMAIN

IP_SUBNET

IP_GATEWAY

IP_PRIMARY_DNS

IP_SECONDARY_DNS

RECORD_TIME

IPX_ADDR

Definition

Describes the system's IPX address. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

IPX_ADDR

NET_NUM

NODE_ADDR

LINK_SPEED

MAX_PACKET_SIZE

RECORD_TIME

MATCHED_SWARE

Definition

Stores the details of a file that matches a software signature. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory software scan.

Columns

COMPUTER_SYS_ID

SWARE_SIG_ID

RECORD_TIME

MOUSE

Definition

Stores the details for one particular type and model of mouse; works with the INST_MOUSE table. The configuration repository contains only one table for a specific device, even if the exact same device is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

MOUSE_ID

BUTTONS

MOUSE_MODEL

MOUSE_TYPE

NATIV_SWARE

Definition

Stores the details for a particular operating system package; works with the INST_NATIVE_SWARE table. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory software scan.

Columns

NATIV_ID

PACKAGE_NAME

PACKAGE_VERS

PUBLISHER

PACKAGE_ID

NET_ADAPTER

Definition

Describes the network adapter installed on a system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

PERM_MAC_ADDR

CURRENT_ADDR

ADAPTER_TYPE

ADAPTER_MODEL

MANUFACTURER

INST_DATE

RECORD_TIME

NW_SERVER

Definition

Describes the settings on a NetWare server. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

NW_DEV_NAME

NW_VERS

NW_SUB_VERS

NW_MAX_CONNS

NW_MAX_VOLS

NW_REVISION_LEVEL

NW_SFT_LEVEL

NW_TTS_LEVEL

NW_MAX_CONNS_USED

NW_ACCOUNTING_VERS

NW_VAP_VERS

NW_QUEING_VERS

NW_PRINTSERVR_VERS

NW_VIRT_CONS

NW_SEC_LEVEL

NW_INET_BRG_SUPP

NW_CLIB_MAJOR_VERS

NW_CLIB_MINOR_VERS

NW_CLIB_REVISION

NW_SER_NUM

RECORD_TIME

NW_VOLS

Definition

Describes a NetWare volume installed on a system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

NWVOL_NAME

NWVOL_TOTAL_BLKS

NWVOL_BLK_SECTORS

NWVOL_AVAIL_BLKS

NWVOL_DIR_SLOTS

NWVOL_AVAIL_SLOTS

NWVOL_IS_REMOVABLE

RECORD_TIME

PC_SYS_PARAMS

Definition

Stores BIOS and other system information for a PC. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

USER_NAME

DOMAIN_NAME

WORKGROUP_NAME

BIOS_ID

BIOS_ID_BYTES

BIOS_DATE

BIOS_STRING

BIOS_MANUFACTURER

BIOS_MODEL

BIOS_SER_NUM

RECORD_TIME

PCI_DEV

Definition

Describes a PCI device installed in or connected to a system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

INST_PCI_ID

PCI_DEV_NAME

RECORD_TIME

PRINTER

Definition

Stores the details for one particular type and model of printer; works with the INST_PRINTER table. The configuration repository contains only one table for a specific device, even if the exact same device is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

PRINTER_ID

PRINTER_MODEL

PROCESSOR

Definition

Stores the details for one particular type and model of processor; works with the INST_PROCESSOR table. The configuration repository contains only one table for a specific device, even if the exact same device is installed on more than one system in the TMR.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

PROCESSOR_ID

MANUFACTURER

PROCESSOR_MODEL

PROCESSOR_FEATURES

MAX_SPEED

CURRENT_SPEED

CHIP_FAMILY

CHIP_MODEL
CHIP_STEPPING
VIRT_MODE_EXT
PAGE_SIZE_EXT
TIME_STAMP_COUNTER
MODEL_SPECIFIC_REG
PHYSICAL_ADDR_EXT
MACHINECHECK_EXCPT
CMPXCHG8B_SUPP
ON_CHIP_APIC
MEM_TYPE_RANGE_REG
PAGE_GLOBAL_ENABLE
MACHINECHECK_ARCH
COND_MOVE_SUPP
MMX_TECHNOLOGY
ON_CHIP_FPU
DEBUG_EXT_PRESENT
FAST_SYS_CALL
PAGE_ATTR_TABLE
PAGE_SIZE_EXT36
SER_NUM_ENABLED
FAST_FLOAT_SAVE
SIMD_EXT_SUPP
NOW_3_D_ARCH

STORAGE_DEV

Definition

Stores the details for a particular storage device installed on the system; works with the HDISK table. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

STORAGE_CLASS

INST_STORAGE_ID

STORAGE_TYPE

MANUFACTURER

MODEL

SER_NUM

HDISK_ID

RECORD_TIME

SWARE_SIG

Definition

Stores the details for a particular software signature; works with the MATCHED_SWARE table and the following views: H_INST_SWARE_VIEW, H_NOSIG_FILES_VIEW, INST_SWARE_VIEW, MATCH_SWARE_VIEW, NOSIG_FILES_VIEW, SWARE_MATCH_CRC32, SWARE_MATCH_MD5, and SWARE_MATCH_QUICK. The configuration repository contains only one table for a specific file, even if the exact same file is installed on more than one system in the TMR.

Comment

Populated by the **winvsig** command and the GUI for Tivoli Inventory, Version 4.0.

Columns

SWARE_SIG_ID

SWARE_NAME

SWARE_SIZE

SWARE_DESC

SWARE_VERS

SIG_SOURCE

SIG_STATUS

CHECKSUM_QUICK

CHECKSUM_CRC32

CHECKSUM_MD5

RECORD_TIME

UNIX_SYS_PARAMS

Definition

Stores UNIX system parameters for a system. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

COMPUTER_SYS_ID

BOOT_TIME

UPTIME

RUN_LEVEL

HOST_NAME

RECORD_TIME

UNMATCHED_FILES

Definition

Stores the details for a file that does not match any software signatures; works with the FILE_DESC and FILE_PATH tables. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory software scan.

Columns

COMPUTER_SYS_ID

FILE_DESC_ID

INST_PATH_ID

CREATED_TIME

MODIFIED_TIME

ACCESSED_TIME

FILE_PERMISSIONS

FILE_OWNER

FILE_GROUP

CHECKSUM_QUICK

CHECKSUM_CRC32

CHECKSUM_MD5

RECORD_TIME

USB_DEV

Definition

Stores the details for one particular type and model of USB device; works with the INST_USB_DEV table. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

USB_ID

USB_VERS

DEV_CLASS

DEV_SUBCLASS

VENDOR_ID

PRODUCT_ID

MANUFACTURER

PRODUCT

NUM_OF_PORTS

DEV_IS_HUB

VID_CARD

Definition

Stores the details for one particular type and model of video card; works with the INST_VID_CARD table. One table exists for each system scanned.

Comment

Populated by a Tivoli Inventory hardware scan.

Columns

VID_CARD_ID

VID_CARD_MODEL

VID_CARD_BIOS

VID_DAC_TYPE

VID_MEM

VID_BIOS_RELDATE

VID_CHIP_TYPE

History Tables

History tables add H_ to the name of the operational data table on which they are based. A history table returns all of the columns returned by its corresponding operational data table plus the RECORD_ACTION and PRFL_ACTION columns.

The following list shows the available history tables:

H_COMPUTER

H_COMPUTER_SYS_MEM

H_INST_HEADER_INFO

H_INST_MOUSE

H_INST_NATIVE_SWARE

H_INST_PARTITION

H_INST_PRINTER

H_INST_PROCESSOR

H_INST_USB_DEV

H_INST_VID_CARD

H_IP_ADDR

H_IPX_ADDR

H_MATCHED_SWARE

H_NET_ADAPTER

H_NW_SERVER

H_NW_VOLS

H_PC_SYS_PARAMS

H_PCI_DEV

H_STORAGE_DEV

H_UNIX_SYS_PARAMS
H_UNMATCHED_FILES

D

Policy

This appendix includes an explanation of policies along with a description of each policy included with Tivoli Inventory.

Policies are rules that govern the resources in policy regions. There are two kinds of policies that set guidelines in a policy region:

- *Default policy*—Enables you to set the defaults for managed resources.
- *Validation policy*—Enables you to create rules that are enforced when administrators add or change managed resources.

See the *Tivoli Management Framework User's Guide* for detailed information about policy and policy regions. This appendix provides information about the default policy methods provided with Tivoli Inventory and an example of how to create new policy methods and policy objects.

The Tivoli Inventory default policy methods described in this appendix set the defaults for **InventoryConfig** managed resources. In other words, the Tivoli Inventory default policy methods determine what options will be set by default when you create a new inventory profile. (Tivoli Inventory does not currently provide validation policies.)

You can change the Tivoli Inventory default policy methods if you want to preset inventory profile properties with specific values. For example, if you want all of your inventory profiles within a particular policy region to update the configuration repository with only the differences since the last scan, you can define a policy method that always sets this option for newly created profiles.

Default Policy Methods

Default policy methods are shell scripts or programs—UNIX scripts (such as Bourne, K, and Perl shell scripts), awk programs, C programs, and so on—that are invoked by Tivoli Inventory when a new inventory profile is created. By creating scripts or programs and replacing the contents of these policy methods, you can control the options that are automatically set in newly created inventory profiles. When creating your own policy method, use the following guidelines:

- It is recommended that you write your policy programs in an interpreter language. Policies are stored in the database, and interpretive programs usually require less space than executable files (compiled programs). Also, an interpretive program can be used across multiple platforms; executable files cannot.
- When Tivoli Inventory invokes a default policy method, the name of the inventory profile being created is passed to the method as the first argument. Tivoli Inventory expects the default policy methods to exit with the code **0**—you must write your policy methods to do so. Reserve other exit codes for hard errors, such as insufficient memory, incorrect usage, and so on.
- Do not use C shell scripts because file descriptors are not handled as needed for the Tivoli Management Framework.

Policy Objects

A policy object is a set of policy methods for a specific resource class. The default policy methods that govern the **InventoryConfig** resource are defined in a default policy object called **BasicInventoryConfig**.

The **BasicInventoryConfig** policy object and the contained policy methods are provided with Tivoli Inventory. You can also create additional policy objects for the **InventoryConfig** resource. Multiple policy objects enable you to define different policies that have the same policy methods.

For example, suppose you have two policy regions called **Data** and **Software**. To create policies that set the initial values for inventory profile names and inventory profile options in each policy region, you can create separate policy objects, such as **DataPolicy** and

SoftwarePolicy. After you define the policies for the methods in each policy object and assign the policy objects to the policy regions, any newly created inventory profiles would adhere to the guidelines you defined.

To define a new policy object and its policy methods, you must perform the following procedures:

- Create a new policy object.
- Replace the contents of the new policy object methods.
- Assign the new policy object to the policy region in which the inventory profiles will reside.

The following sections provide detailed instructions on each of these procedures. See the *Tivoli Management Framework User's Guide* for detailed information about checking policy in a policy region.

Creating a New Policy Object

To define different policies for multiple policy regions, you must create new policy objects. If you do not define policy for a policy region, Tivoli Inventory uses the **BasicInventoryConfig** policy object and its policy methods by default.

The following table provides the authorization roles required to create a new policy object:

Operation	Context	Role
Create a new policy object	TMR	**senior** or **super**

You must use the command line to create a new policy object.

Enter the following syntax for the **wcrtpol** command to create a policy default object:

```
wcrtpol -d InventoryConfig DataPolicy
```

where:

-d

Creates a policy default object.

InventoryConfig

Specifies **InventoryConfig** as the resource type of the new policy default object.

DataPolicy

Specifies **DataPolicy** as the name of the new policy default object.

A new policy object inherits its policy methods from an existing policy object. After you create a new policy object, you can view the existing policy methods to validate inventory profile properties or operations for a particular policy region using the following commands:

wlspolm

Lists the policy methods for the specified resource. You can list the default or validation policy methods with this command.

wgetpolm

Retrieves the contents of the specified default or validation policy method. Policy methods are inherited from an existing **InventoryConfig** policy object.

For more information about the **wcrtpol**, **wlspolm**, and **wgetpolm** **c**ommands, see the *Tivoli Management Framework Reference Manual*.

Replacing the Contents of a Policy Method

To define different policies from those inherited by the parent policy object, you must create a script or program and replace the existing policy method with it.

The following table provides the authorization roles required to replace the contents of a policy method:

Operation	Context	Role
Replace the content of a policy method	TMR	**senior** or **super**

You must use the command line to replace the content of a policy method.

Enter the following syntax for the **wputpolm** command to replace the contents of the **ic_def_global_logfile_path** policy method with the contents of the **Data_logfile_def.sh** script:

```
wputpolm -d InventoryConfig DataPolicy \
ic_def_global_logfile_path < Data_logfile_def.sh
```

where:

-d

Specifies that the method is a policy default method.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

DataPolicy

Specifies **DataPolicy** as the policy object that contains the default policy method being replaced.

ic_def_global_logfile_path

Replaces the contents of the **ic_def_global_logfile_path** default policy method.

< Data_logfile_def.sh

Redirects the **Data_logfile_def.sh** script to the command. The contents of this file replace the existing contents of the **ic_def_global_logfile_path** policy method.

For more information about the **wputpolm** command, see the *Tivoli Management Framework Reference Manual*.

Assigning Policy to a Policy Region

To change the default policy for a policy region, you must assign policy to the policy region after you have created a new policy object and replaced policy methods.

The following table provides the authorization roles required to assign policy to a policy region:

Operation	Context	Role
Assign policy to a policy region	Policy Region	**senior** or **super**

You can use either the Tivoli desktop or the command line to assign policy to a policy region. See the *Tivoli Management Framework User's Guide* for instructions about how to use the Tivoli desktop.

To use the **wsetpr** command to change the default policy in the **Dev** policy region to those methods defined in the **DataPolicy** policy object, enter the following command:

```
wsetpr -d DataPolicy InventoryConfig @PolicyRegion:Dev
```

where:

-d DataPolicy

Changes the default policy to that defined in the **DataPolicy** object.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

@PolicyRegion:Dev

Specifies **Dev** as the policy region for which to assign the policy.

For more information about the **wsetpr** command, see the *Tivoli Management Framework Reference Manual*.

Example—Setting a Default Policy Method

The following example provides the command line solution of how to set the policy default for the default list of file extensions to be matched,

and how to create and assign policy to a new policy object. Complete the following steps:

1. List the policy default methods for the **InventoryConfig** class by entering the following command from a root prompt:

   ```
   wlspolm -d InventoryConfig
   ```

   where:

   -d

   Lists the policy default methods for the **InventoryConfig** resource type.

   InventoryConfig

   Specifies **InventoryConfig** as the resource whose policy methods are to be listed.

   The following default policies are returned:

   ```
   ic_def_global_action
   ic_def_global_ep_timeout
   ic_def_global_logfile_host
   ic_def_global_logfile_path
   ic_def_global_notice_interval
   ic_def_global_notice_location
   ic_def_global_notice_type
   ic_def_global_update_replace
   ic_def_pc_custom_mifs
   ic_def_pc_custom_script
   ic_def_pc_exclude_dirs
   ic_def_pc_extensions
   ic_def_pc_hw_gran
   ic_def_pc_hw_outfile
   ic_def_pc_hw_scan
   ic_def_pc_include_dirs
   ic_def_pc_sw_crc
   ic_def_pc_sw_flags
   ic_def_pc_sw_outfile
   ic_def_pc_sw_scan
   ic_def_unix_custom_mifs
   ic_def_unix_exclude_dirs
   ic_def_unix_extensions
   ic_def_unix_hw_gran
   ic_def_unix_hw_outfile
   ```

```
ic_def_unix_hw_scan
ic_def_unix_include_dirs
ic_def_unix_sw_crc
ic_def_unix_sw_flags
ic_def_unix_sw_outfile
ic_def_unix_sw_scan
```

2. List the policy default objects that exist for the **InventoryConfig** class. The Tivoli Management Framework supports multiple policy default objects, which means that you can have one set of policy defaults in policy region **X** and a different set in policy region **Y**. Use the following command to list the policy default objects for Tivoli Inventory:

   ```
   wlspol -d InventoryConfig
   ```

   where:

   -d

   Lists the policy default objects for the **InventoryConfig** resource type.

   InventoryConfig

   Specifies **InventoryConfig** as the resource whose policy objects are to be listed.

   This command returns only the **BasicInventoryConfig** object if you have not created additional policy default objects.

3. Extract the current contents of the **ic_def_pc_extensions** policy default method to make sure that another administrator has not modified it.

   ```
   wgetpolm -d InventoryConfig BasicInventoryConfig \
   ic_def_pc_extensions
   ```

   where:

   -d

   Lists the contents of a policy default method.

   InventoryConfig

   Specifies **InventoryConfig** as the resource whose policy is to be returned.

BasicInventoryConfig

Specifies **BasicInventoryConfig** as the policy object whose policy is to be returned.

ic_def_pc_extensions

Specifies the policy method whose contents are to be returned.

The contents of this policy method are output to standard output by default. This command returns the following output:

```
#!/bin/sh
#  Component Name:


#  $Date: 2000/10/03 15:00:00 $
#
#  $Source: ic_def_pc_extensions,v $
#
# $Id: ic_def_pc_extensions,v 1.0 2000/10/03 15:00:00 plw Exp
$
#
#  Description:
#
#  Copyright 2000 by Tivoli Systems, Inc.  An IBM Company
#  Unpublished Work
#  All Rights Reserved
#  Licensed Material - Property of TIVOLI Systems, Inc.  An
IBM Company


EXCLUDE=0
INCLUDE=1


# MUST set Include or Exclude first before listing extensions.
# If no extensions are listed then just select on Include or
Exclude.  It will
# not matter.
echo $INCLUDE
```

```
# PC extensions MUST be in uppercase.  No validation will be
done.
echo "*.EXE
*.DLL
*.COM
*.NLM"


exit 0
```

4. Create a script that changes the extension list to an EXCLUDE list, and sets the list of extensions to exclude. The following script, called /tmp/list.sh, accomplishes this:

```
#!/bin/sh
# Custom policy

EXCLUDE=0
INCLUDE=1

# MUST set Include or Exclude first before listing extensions.
# If no extensions are listed then just select on Include or
Exclude.  It will
# not matter.
echo $EXCLUDE


echo "*.BAK
*.TMP
*.SYS
*.INI"

exit 0
```

5. Replace the contents of the **ic_def_pc_extensions** policy method with the new script using the following command:

```
wputpolm -d InventoryConfig BasicInventoryConfig \
ic_def_pc_extensions </tmp/list.sh
```

where:

-d

Specifies that the **ic_def_pc_extensions** policy method is a default policy method.

InventoryConfig

Specifies **InventoryConfig** as the resource for which the policy is set.

BasicInventoryConfig

Specifies **BasicInventoryConfig** as the policy object for which the policy is set.

ic_def_pc_extensions

Specifies **ic_def_pc_extensions** as the policy method whose contents are to be replaced.

</tmp/list.sh

Redirects the /tmp/list.sh script to the command. This command reads its input from standard input.

6. Associate the new policy method in the **BasicInventoryConfig** policy object with the **Dev** policy region:

```
wsetpr -d BasicInventoryConfig InventoryConfig \
@PolicyRegion:Dev
```

where:

-d BasicInventoryConfig

Changes the default policy to that defined in the **BasicInventoryConfig** object.

InventoryConfig

Specifies **InventoryConfig** as the resource type for which the policy is defined.

@PolicyRegion:Dev

Specifies **Dev** as the policy region for which to assign the policy.

After setting the policy, every inventory profile created in policy regions whose default policy for the **InventoryConfig** resource type is set to **BasicInventoryConfig** will have the list of file extensions set as specified in the sample script.

Policy Methods

The following default policy methods enable you to control the default settings in **InventoryConfig** managed resources. These methods can differ among policy regions. That is, one policy region could have one

set of default policy methods and another policy region could have another.

Note: The query default policy methods are also listed here, though they are packaged with the Tivoli Management Framework.

Policy methods must return a zero for the exit code if the method was successful. If the method fails, a non-zero exit code is returned. Information that the method returns must be written as a string to stdout.

ic_def_global_action

Specifies whether to distribute only the configuration file, or to distribute the configuration file and also perform a scan.

SYNOPSIS

ic_def_global_action

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies whether an inventory profile distributes only the configuration file, or distributes the configuration file and also performs a scan. You might want to only distribute the configuration file if you are performing a disconnected scan.

Permissible values for this method are as follows:

16	Specifies to only distribute the configuration file.
17	Specifies to distribute the configuration file and perform a scan. This is the default value.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_ep_timeout

Specifies the default time period, in seconds, after which a scan of an endpoint times out.

SYNOPSIS

ic_def_global_ep_timeout

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies the default value of the endpt_time_out attribute. Tivoli Inventory times the duration of a scan on each endpoint. If the scan does not complete (the endpoint does not return status for the scan) before the time period set by this method, the endpoint is timed out, and Tivoli Inventory indicates that the scan has failed for that endpoint.

Permissible values for this method are a decimal number greater than 0. The default value is 1800 seconds.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_logfile_host

Specifies the name of the managed node on which the log file for Tivoli Inventory status information is to be stored.

SYNOPSIS

ic_def_global_logfile_host

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies the name of the managed node on which to write the log file for Tivoli Inventory status information. This method is used only if an inventory profile has been configured to write the status information to a log file. If no managed node is specified, the default option is the managed node on which the inventory data handler is installed. If a name is specified, the name must match the name of a valid managed node. An example is as follows:

```
echo "rainier"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_logfile_path

Specifies the path and name for the log file for Tivoli Inventory status information.

SYNOPSIS

ic_def_global_logfile_path

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies the path and name for the log file that holds Tivoli Inventory status information. This method is used only if an inventory profile has been configured to write the status information to a log file. If no value is specified in this method, the file is named inv_scan_*n*, where *n* is the ID number of the scan. The file is placed in the temporary directory of the managed node specified by ic_def_global_logfile_host. This temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows: c:\temp

An example is as follows:

```
 echo "/tmp/pc_scan_hw.log"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_notice_interval

Specifies when notification is sent for a scan that is configured to report status.

SYNOPSIS

ic_def_global_notice_interval

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies when notification should be sent for a scan. The options are to send notification as each target completes, to send notification at periodic intervals, or to send notification only when all targets have completed.

Permissible values for this method are as follows:

1	Send notification as each target completes.
16	Send notification after a specific time period, and/or send notification each time a specified number of targets complete. The time period and number of targets complete is based on the settings for the inventory data handler.
256	Send notification only after all targets are complete. This is the default value.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_notice_location

Specifies the locations to which notifications are sent.

SYNOPSIS

ic_def_global_notice_location

RESOURCE

InventoryConfig

DESCRIPTION

This method indicates where status information for a profile should be written during a scan. You can specify one or more of the following options:

Permissible values for this method are as follows:

1	Write status information to the inventory notice group. This is the default value.
16	Write status information to a log file.
256	Write status information to the Tivoli Enterprise Console (TEC).

No value, or a value of 0, indicates to not write any status information. You can specify that you want to write to multiple locations by adding the values for the writing status information. The following example writes status information to the inventory notice group and TEC:

```
echo 257
 or
expr 256 + 1
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_notice_type

Specifies the types of target scans for which notification should be sent.

SYNOPSIS

ic_def_global_notice_type

RESOURCE

InventoryConfig

DESCRIPTION

This method returns a numeric value indicating the circumstances for which a notice is generated.

Permissible values for this method are as follows:

1 Send notification for targets that succeed.

16 Send notification for targets that fail.

You can specify to send notices for both successful and failed targets with the value 17 (16 + 1). You can specify to not send notices by specifying a value of 0. The default value is 17.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_global_update_replace

Specifies whether Tivoli Inventory replaces all data or replaces only differences when the configuration repository is updated with the results of a scan.

SYNOPSIS

ic_def_global_update_replace

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies whether Tivoli Inventory replaces all data in the configuration repository during a scan, or compares the current scan data to the MIF file generated by the previous scan and replaces only differences. If an endpoint does not yet have scan data in the database, all data for that endpoint is sent to the configuration repository.

Permissible values for this method are as follows:

1 Specifies to compare the results of a scan with the results of the previous scan and save the differences in the configuration repository. This is the default option.

16 Specifies to save the current scan results in the configuration repository.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_custom_mifs

Specifies the path and file name of custom MIF files that are read during the scan of a PC.

SYNOPSIS

ip_def_custom_mif_file

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the list of custom MIF files that Tivoli Inventory reads from PC target machines. By default, no custom MIF files are read. If this method writes out more than one custom MIF file, each file name must be written out on a new line. Directory names must be delimited using forward slashes (/) as in the following example:

```
echo "useradd.mif
C:/scandata/data.mif"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_custom_script

Specifies the default scripts to be run during the scan of a PC.

SYNOPSIS

ic_def_pc_custom_script

RESOURCE

InventoryConfig

DESCRIPTION

This method specifies the custom scripts to be run during a PC scan. Scripts are run after the Tivoli scanner completes but before the MIF files are read. Scripts specified by this method are usually .BAT or .CMD scripts. An example follows:

```
echo "C:/MYSCAN.EXE
COPY C:/MYSCAN.MIF D:/TIVOLI/USERADD.MIF
```

By default, no custom scripts are run.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_exclude_dirs

Specifies the default list of directories that are excluded during a software scan of a PC.

SYNOPSIS

ic_def_pc_exclude_dirs

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the default list of directories that an inventory profile does not scan during a software scan of a PC. The method should write a list of directory names, each on a separate line, to stdout. By default, the */TMP, */TEMP, /RECYCLED/, and /RECYCLER/ directories are not scanned.

Directory names must be uppercase and delimited with forward slashes. An example follows:

```
echo "*/CACHE
*/BACKUP"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_extensions

Specifies a list of files or file types to either include or exclude during a software scan of PCs.

SYNOPSIS

ic_def_pc_extensions

RESOURCE

InventoryConfig

DESCRIPTION

This method provides a list of file names (such as file.exe), file types (such as *.exe), or both. The method generates a value indicating whether the files or file types are to be included or excluded during a software scan of a PC.

By default, the *.exe, *.dll, *.com, *.nlm, and *.sig file types are included in software scans of a PC.

The first line of method output is one of the following options:

```
0     Exclude the list of extensions from a PC software scan
1     Include the list of extensions from a PC software scan
```

Following lines of output are a list of file names or file types, separated by a new line, as in the following example:

```
echo 1
echo "*.EXE
*.DLL
*.DRV
*.NLM
CONFIG.SYS"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_hw_gran

Indicates the hardware components that are scanned for on a PC.

SYNOPSIS

ic_def_pc_hw_gran

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a list of the hardware components that can be scanned for during a PC scan and assigns a value for each component. The value indicates whether or not to collect information for that component. The following is the contents of the method provided by Tivoli. The list contains all the potential component names that you can use in your method.

```
$/bin/sh
OFF=0
ON=1
echo "Processor $ON"
echo "Memory $ON"
echo "Operating System $ON"
echo "Storage $ON"
echo "IP Address $ON"
echo "Network Adapter $ON"
echo "Partition $ON"
echo "Pointing Device $ON"
echo "Keyboard $ON"
echo "IPX Address $ON"
echo "Video $ON"
echo "Printer $ON"
echo "USB Device $ON"
exit 0
```

By default, all hardware components in this list are scanned for.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_hw_outfile

Specifies the Tivoli MIF files that are retrieved from PCs during a hardware scan.

SYNOPSIS

ic_def_pc_hw_outfile

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from a PC endpoint and sent to the configuration repository during a hardware scan. Permissible values for this method are as follows:

0	Do not return any of the MIF files during a hardware scan.
1	Return the hardware MIF file produced by the Tivoli scanner. This is the default option.
16	Return the hardware MIF file produced by the DMI scanner.

You can specify that you want both MIF files returned by adding the values for the Tivoli hardware scanner and the DMI scanner. For example:

```
echo 17
 or
expr 16 + 1
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_hw_scan

Specifies which hardware scanners to run when scanning PC endpoints.

SYNOPSIS

ic_def_pc_hw_scan

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value indicating which hardware scanners to run during a scan of PCs. Permissible values for this method are as follows:

0	Do not run any hardware scanners.
1	Run the Tivoli scanner. This is the default option.
16	Run the DMI scanner.

You can run both the Tivoli and DMI scanners by adding the values for the Tivoli scanner and the DMI scanner. For example:

```
echo 17
 or
expr 16 + 1
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_include_dirs

Specifies the default list of directories that are included during a software scan of a PC.

SYNOPSIS

ic_def_pc_include_dirs

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the default list of directories that are scanned during a software scan of a PC. The method should write a list of directory names to stdout, each directory separated by a new line. PC directory names must be uppercase, and directories must be delimited by forward slashes (/). For example:

```
echo "*/PROGRAM_FILES
*/APPLICATIONS"
exit 0.
```

By default, no directories are specified (all directories are scanned).

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_sw_crc

Specifies which cyclic redundancy check (CRC) value, if any, is obtained for files during a software scan of PCs.

SYNOPSIS

ic_def_pc_sw_crc

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which CRC value is obtained for files during a PC software scan. Only one type of CRC value can be collected during a software scan. The following values are permissible:

0	Do not compute a CRC for files. This is the default option.
1	Produce a 4-byte checksum for the first 1024 bytes of the file.
16	Produce a 4-byte checksum for all the bytes in the file.
256	Produce a 16-byte MD5 checksum for a file.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_sw_flags

Specifies which special options are in effect during software scans of PCs.

SYNOPSIS

ic_def_pc_sw_flags

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which special options are in effect during a software scan of a PC. The options available indicate whether to scan for executables only, and whether to apply a custom filter to the basic information software scan. The permissible option values are as follows:

0	No special PC software scan options. This is the default option.
1	Scan only for executable files.
16	Apply a custom filter to the basic information software. scan.

You can specify more than one option by generating the sum of the desired options. For example, you can specify to scan only for executables and apply a custom filter as follows:

```
echo 17
 or
expr 1 + 16
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_sw_outfile

Specifies the Tivoli MIF files that are retrieved from PCs during a software scan.

SYNOPSIS

ic_def_pc_sw_outfile

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from a PC endpoint and sent to the configuration repository during a software scan. Permissible values for this method are as follows:

0	Do not return any of the MIF files.
1	Return the MIF file that contains basic file information.
16	Return the MIF file that contains file header information.
256	Return the MIF file that contains registry information.
4096	Return the software MIF file produced by software signature matching.

You can specify that you want multiple MIF files returned by adding the values for desired the MIF files. For example, you can specify to return the basic file information, the file header information, and the PC registry information as follows:

```
echo 273
 or
expr 256 + 16 + 1
```

The default value is 272 (256 + 16).

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_pc_sw_scan

Specifies which software scanners to run when scanning PC endpoints.

SYNOPSIS

ic_def_pc_sw_scan

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value indicating which software scanners to run on a PC during a software scan. Permissible values are as follows:

0	Do not run any software scanners.
1	Scan for basic file information.
16	Scan for file header information.
256	Scan the PC registry. This is the default option.
4096	Scan for file signature information.

You can specify that you want to run multiple scanners by adding the values for the scans you want. For example, you can run scanners for the PC registry, file information, and basic file information as follows:

```
echo 273
 or
expr 256 + 16 + 1
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_custom_mifs

Specifies the path and file names of custom MIF files that are read during the scan of UNIX systems.

SYNOPSIS

ic_def_unix_custom_mifs

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the list of custom MIF files that Tivoli Inventory retrieves from UNIX target machines. By default, no custom MIF files are retrieved. If you list more than one custom MIF file, the file names in the output should be separated by new lines. For example:

```
echo "useradd.mif
/usr/apps/custom.mif"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_custom_script

Specifies a user-defined script to be run during the scan of a UNIX system.

SYNOPSIS

ic_def_unix_custom_script

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a script to be run during a scan (after the scan completes but before the MIF files are read). On UNIX systems, this script is normally a .sh script. For example:

```
echo "#!/bin/sh
/usr/local/bin/myscan.sh
cp /tmp/scandata.mif /tivoli/useradd.mif
"
```

By default, no scripts are run.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_exclude_dirs

Specifies the default list of directories that are excluded during a software scan of a UNIX system.

SYNOPSIS

ic_def_unix_exclude_dirs

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the default list of directories that an inventory profile will not scan during a profile distribution to a UNIX system. By default, the */tmp directory is excluded.

The method must write a list of directory names to stdout, each separated by a new line. For example:

```
echo "*/tmp
*/temp/"
exit 0
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_extensions

Specifies a list of file names or file types to either include or exclude during a software scan of a UNIX system.

SYNOPSIS

ic_def_unix_extensions

RESOURCE

InventoryConfig

DESCRIPTION

This method provides a list of file names (such as readme.txt), file types (such as *.txt), or both. The method generates a value indicating whether the file names and file types are to be included or excluded in a software scan of a UNIX system.

The first line of method output is one of the following options:

```
0     Exclude the list of extensions from a UNIX software scan
1     Include the list of extensions in a UNIX software scan
```

Following lines of output are a list of file names or file types, separated by a new line, as in the following example:

```
echo 1
echo "*.txt
myfile
*.TXT"
exit 0
```

By default, no file names or file types are specified (all files are included in the scan).

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_hw_gran

Indicates the hardware components that are scanned for on a UNIX system.

SYNOPSIS

ic_def_unix_hw_gran

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a list of the hardware components that can be scanned for during a UNIX scan and assigns a value for each component. The value indicates whether or not to collect information for that component. The following is the contents of the method provided by Tivoli. The list contains all the potential component names that you can use in your method.

```
#!/bin/sh
OFF=0
ON=1
echo "Processor $ON"
echo "Memory $ON"
echo "Operating System $ON"
echo "Storage $ON"
echo "IP Address $ON"
echo "Network Adapter $ON"
echo "Partition $ON"
echo "Pointing Device $ON"
echo "Keyboard $ON"
echo "Unix System Params $ON"
exit 0
```

By default, all components in this list are scanned for.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_hw_outfile

Specifies the Tivoli MIF files that are retrieved from UNIX systems during a hardware scan.

SYNOPSIS

ic_def_unix_hw_outfile

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates whether the MIF that contains the hardware scan data is retrieved from a UNIX endpoint and sent to the configuration repository during a hardware scan. Permissible values for this method are as follows:

0 Do not return the hardware MIF file during a hardware scan.

1 Return the hardware MIF file. This is the default option.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_hw_scan

Specifies whether to perform a hardware scan on UNIX endpoints.

SYNOPSIS

ic_def_unix_hw_scan

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value indicating whether to perform a hardware scan on UNIX endpoints. The following are the permissible values:

0	Do not perform a hardware scan on UNIX endpoints.
1	Perform a hardware scan on UNIX endpoints. This is the default option.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_include_dirs

Specifies the default list of directories that are included during a software scan of a UNIX system.

SYNOPSIS

ic_def_unix_include_dirs

RESOURCE

InventoryConfig

DESCRIPTION

This method generates the default list of directories that are scanned during a software scan of a UNIX system. The method should write a list of directory names to stdout, each directory separated by a new line. Directories must be delimited by forward slashes (/). For example:

```
echo "/usr/local/bin
*/bin"
exit 0
```

By default, no directories are specified (all directories are scanned).

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_sw_crc

Specifies which cyclic redundancy check (CRC) value, if any, is obtained for files during a software scan of UNIX systems.

SYNOPSIS

ic_def_unix_sw_crc

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which CRC value is obtained for files during a UNIX software scan. Only one type of CRC value can be collected during a software scan. The following values are permissible:

0	Do not compute a CRC for files. This is the default option.
1	Produce a 4-byte checksum for the first 1024 bytes of the file.
16	Produce a 4-byte checksum for all the bytes in the file.
256	Produce a 16-byte MD5 checksum for a file.

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_sw_flags

Specifies which special options are in effect during software scans of UNIX systems.

SYNOPSIS

ic_def_unix_sw_flags

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which special options are in effect during a software scan of a UNIX system. The options available indicate whether to scan for executables only, and whether to apply a custom filter to the basic information software scan. The permissible option values are as follows:

0	No special UNIX software scan options.
1	Scan only for executable files. This is the default option.
16	Apply a custom filter to the basic information software. scan.

You can specify more than one option by generating the sum of the desired options. For example, you can specify to scan only for executables and apply a custom filter as follows:

```
echo 17
 or
expr 1 + 16
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_sw_outfile

Specifies the Tivoli MIF files that are retrieved from UNIX systems during a software scan.

SYNOPSIS

ic_def_unix_sw_outfile

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value that indicates which MIF files, and therefore what scan data, is retrieved from a UNIX endpoint and sent to the configuration repository during a software scan. Permissible values for this method are as follows:

0	Do not return any of the MIF files.
1	Return the MIF file that contains basic file information.
256	Return the MIF file that contains installed product and patch information. This is the default option.
4096	Return the software MIF file produced by software signature matching.

You can specify that you want multiple MIF files returned by adding the values for desired the MIF files. For example, you can specify to return the basic file information and the software signature information as follows:

```
echo 4097
 or
expr 4096 + 1
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

ic_def_unix_sw_scan

Specifies which software scanners to run when scanning UNIX endpoints.

SYNOPSIS

ic_def_unix_sw_scan

RESOURCE

InventoryConfig

DESCRIPTION

This method generates a decimal value indicating which software scanners to run on a UNIX system during a software scan. Permissible values are as follows:

0	Do not run any software scanners.
1	Scan for basic file information.
256	Scan for installed product and patch information. This is the default option.
4096	Scan for file signature information.

You can specify that you want to run multiple scanners by adding the values for the scans you want. For example, to scan for basic file information and installed product and patch information:

```
echo 257
 or
expr 256 + 1
```

Tivoli Inventory policy methods exit with **0** if they execute successfully. Policy methods exit with a non-zero exit code if a failure occurs. Information that the method returns must be written as a string to stdout.

E

Troubleshooting

This appendix describes files and procedures that can help you research and resolve problems that you may encounter while using Tivoli Inventory. In general, you should check the Tivoli Inventory notice group for troubleshooting information before consulting the sources described in this appendix. See the *Tivoli Management Framework User's Guide* for information about notice groups.

Configuration Repository Schema

To troubleshoot problems during the creation of the configuration repository, check the log file named inv_*rdbms_type*_admin.log, where *rdbms_type* is the type of RDBMS you are using for the configuration repository. This variable can be one of the following values: db2, ms, ora, or syb. The success or failure of the SQL statements are logged in this log file. Check your RDBMS documentation to determine the location of this log file.

DMI

This section contains information about troubleshooting DMI scans.

Adding a DMI Layer

Before you can scan for DMI data, you must configure DMI scanning options. As a part of this process, you must add each type of DMI layer you want to scan to the table in the DMI Scanner Configuration dialog

box. Before you attempt to add a DMI layer, make sure that you have distributed an inventory profile at least once to the endpoint on which the DMI layer is installed. An inventory profile distributes the following files to an endpoint: dmiscan.exe, wcdmi.dll, and wdmiutil.dll. These files must be present on the endpoint before you can retrieve the DMI layer.

When you attempt to add a DMI layer to the table in the DMI Scanner Configuration dialog box, you might see one of the errors in the following list. Troubleshooting information is provided for each error.

- **Could not find endpoint**.

  To troubleshoot this error message, run the following command to ensure that the endpoint is available:

  ```
  wadminep endpoint_name view_log_file
  ```

  where:

  endpoint_name Specifies the name of the endpoint.

  view_log_file Displays a log file for the endpoint.

  If you cannot view the log file, the endpoint is not available. Troubleshoot the endpoint and the connection between the endpoint and its gateway.

- **Could not examine DMI layer**.

  A DMI layer exists on the specified endpoint, but the examination of the DMI layer timed out. Retry the operation later.

- **No DMI layer exists on this endpoint**.

  The endpoint you specified does not have a DMI layer. Install a DMI layer on this endpoint, or specify a different endpoint that has the same type of DMI layer.

- **Could not retrieve DMI data**.

  There is a communication problem between the endpoint and the gateway to which it belongs. To troubleshoot this error message, run the following command:

  ```
  wadminep endpoint_name view_log_file
  ```

  where:

  endpoint_name Specifies the name of the endpoint.

view_log_file Displays a log file for the endpoint.

If you cannot view the log file, the endpoint is not available. Troubleshoot the endpoint and the connection between the endpoint and its gateway.

- **DMI data format is not recognized.**

  To troubleshoot this error message, perform the following tasks:

  - Specify a different endpoint that has the same type of DMI layer.
  - Examine the DMI layer on the endpoint using a DMI browser such as Intel DMI Explorer to determine whether the data is corrupted or contains an unrecognized value.
  - Reinstall the DMI layer on the endpoint.

For more information about using the DMI Scanner Configuration dialog box and configuring DMI scan options, see the Tivoli Inventory online help.

DMI Scans

If a DMI scan fails, perform the following tasks to troubleshoot the scan:

- Verify that a DMI layer exists on the endpoint for which the scan failed. For PCs that do not contain a DMI layer, a DMI scan will return a value of `NO_DATA` for all attributes. To check whether a DMI layer is installed on an endpoint, follow the procedure to add the DMI layer from the endpoint to the list of DMI layers in the DMI Scanner Configuration dialog box. If no DMI layer is installed on that endpoint, the following message is displayed:

  ```
  No DMI layer exists on this endpoint.
  ```

  For more information about the procedure to add a DMI layer, see the Tivoli Inventory online help.

- Make sure that the file dmi.ini was distributed to the endpoint. This file contains the DMI scan settings. The inventory profile must successfully distribute this file to the endpoint for a DMI scan to complete.

- Run the DMI scan on the endpoint from the command line. The DMI scanner is located in $LCF_INSTDIR/inv/SCAN, where $LCF_INSTDIR is the directory in which the endpoint is installed. Run the DMI scan using the following command:

  ```
  dmiscan -i dmi.ini -o dmiscan.mif
  ```

DMI Table and Column Names

The dmi_*RDBMS_type*_schema.sql script creates custom tables in the configuration repository for DMI scan information. The default name for each custom table is the DMI group name truncated to 18 characters, with underscores (_) substituted for blank spaces. The default names for columns in each table are the DMI attribute names truncated to 18 characters, with underscores substituted for blank spaces.

Because these names are generated automatically, it is possible that the script will use SQL-92 reserved words for table and column names. If this happens, you must rename the table or column. For more information about SQL-92 reserved words, consult a database administrator or SQL documentation.

It is also possible that the script will use invalid characters in table or column names. You can name tables and columns using the following characters:

- Lowercase a through z
- Uppercase A through Z
- Numeric characters 0 through 9
- Underscores

You cannot use spaces. Table and column names must start with a letter.

Endpoint-Initiated Scans

To troubleshoot an endpoint-initiated scan, it is helpful to understand how it differs from scans initiated by a profile distribution. A scan initiated by a profile distribution is composed of the following steps:

- The endpoint is scanned, and then the scan data is saved on the endpoint in a Management Information Format (MIF) file. The MIF file is named *scan_type*.mif, where *scan_type* is the type of

scan. For example, the Tivoli hardware scanner generates tivhscan.mif, and the DMI scanner generates dmiscan.mif.

- Assuming the inventory profile is configured to send the scan results to the configuration repository, the endpoint method parses the MIF file and renames it with a .BK1 extension. The data parsed from the MIF file is written to a .dat file (an encoded and compressed binary file). The .dat file is named inv*scanID*.dat, where *scanID* is the numerical value that identifies the scan.
- The .dat file is sent through the collector hierarchy to the configuration repository.

Endpoint-initiated scans are controlled using the **wepscan** command. With this type of scan, the MIF file is not renamed with a .BK1 extension until you send the data to the configuration repository using the **wepscan** command and **–s** option. In other words, if you run two endpoint-initiated scans consecutively without sending the data to the configuration repository, the second scan overwrites the MIF file generated by the first scan. The data from the first scan is never collected. This is done to ensure that the .BK1 file always contains the data that was last sent to the configuration repository (not necessarily data from the last scan that was run). This design ensures that, if you choose to update with differences, the scan always updates the database with the correct information. (A scan that updates with differences compares the MIF file of the current scan with the .BK1 scan and sends only new or changed data to the configuration repository.)

If you run the **wepscan** command with the **–d** option, the command creates log files called sa_results.log and sa_config.log. If you run the **wepscan** command with the **–c** option, the command creates a log file called sa_config.log. These log files are stored in the directory from which you run the **wepscan** command. The log files contain the following information:

sa_results.log The contents of the DAT file. Check this file to see what data was collected by the scan. This data will be sent to the configuration repository when the scan data is collected.

sa_config.log The configuration of the profile. Check this file to see what software and hardware information the profile is

configured to collect, and other profile configuration information.

Before you run an endpoint-initiated scan, make sure that you have performed the following tasks:

- Distributed an inventory profile to the endpoint. Doing this installs the configuration file, config.dmp, on the endpoint. This file must be present for the endpoint-initiated scan to succeed. To verify that config.dmp is installed on the endpoint, check the inv/SCAN directory in the directory in which the endpoint is installed.
- Set up your environment to access a DLL file needed by the command. For more information about setting up your environment to use **wepscan**, see "wepscan" on page B-31.

Graphical User Interface

This section contains instructions for troubleshooting the Tivoli Inventory graphical user interface (GUI).

System Requirements

The GUI for Tivoli Inventory, Version 4.0, requires Java Runtime Environment (JRE) 1.2.2, which can be installed as a product from the Tivoli Management Framework CD. Not all operating systems that are supported by Tivoli Inventory are compatible with JRE 1.2.2. For more information, see the *Tivoli Inventory Release Notes, Version 4.0* and the documentation for your operating system.

Enabling the Log File

You can configure the Tivoli Inventory GUI to write messages to a log file. You can then use the log file to debug the GUI. This log file is named DEBUG3 and is stored in the $DBDIR directory. To enable the log file for the GUI, use following procedure:

1. On the system on which you are using the GUI, open the inv_gui.sh file in a text editor. This file is located in $BINDIR/TME/INVENTORY.

2. At the bottom of the inv_gui.sh file, locate the text similar to the following example:

```
if [ "$INTERP" = "w32-ix86" ]
then
#  use for debugging
#  eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND" >>
$DBDIR/DEBUG3 2>&1
eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND"
else
#   eval "$COMMAND" >> $DBDIR/DEBUG3 2>&1
eval "$COMMAND"
fi
```

3. Delete the comment tag from the line that creates the DEBUG3 log file, and comment out the line that does not.

   - For Windows platforms, locate the following lines:

     ```
     #  eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND" >>
     $DBDIR/DEBUG3 2>&1
     eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND"
     ```

     Edit the lines as follows:

     ```
     eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND" >>
     $DBDIR/DEBUG3 2>&1
     # eval "$INSTALL_DIR/bin/wrunui.exe $COMMAND"
     ```

   - Or, for UNIX platforms, locate the following lines:

     ```
     #   eval "$COMMAND" >> $DBDIR/DEBUG3 2>&1
     eval "$COMMAND"
     ```

     Edit the lines as follows:

     ```
     eval "$COMMAND" >> $DBDIR/DEBUG3 2>&1
     # eval "$COMMAND"
     ```

Enabling the GUI

To run the Tivoli Inventory GUI on a UNIX system, you must enable connections to the X Window System on the UNIX system. This is necessary even if the console runs on the same machine as the X Window System display. To do this, follow these steps:

1. Set the DISPLAY environment variable to the X Window System display on which to display the Distribution Status console.

For example, to open the console on the X Window System display named zeus:0.0, a Bourne or Korn shell user would enter the following commands:

```
DISPLAY=zeus:0.0
export DISPLAY
```

In some instances, you might need to specify an IP address for the DISPLAY variable rather than the system name.

2. Enable remote connections to the X Window System. For example, to start the Distribution Status console on the zeus display, enter the following command:

```
xhost +zeus
```

Ensure that remote logins are enabled on the system on which you start the console. To ensure that remote logins are enabled, enter the following command:

```
odadmin odinfo
```

The following message is displayed:

```
Remote client login allowed = value
```

where *value* is either **TRUE** or **FALSE**.

If **FALSE**, enter the following command to enable remote client logins:

```
odadmin set_allow_rconnect TRUE
```

Scalable Collection Service

This section contains information about troubleshooting Scalable Collection Service (SCS).

Log and Data Files

The following sections describe SCS-related log and data files. These files reside in one of the following directories, unless a different directory is specified:

- On the inventory data handler—$DBDIR/inventory/mc/*file_name*
- On all other collectors—$DBDIR/mcollect/*file_name*

where:

$DBDIR — Is the Tivoli object database directory on the managed node

file_name Represents one of the file names listed in the following sections

Queue Data Files

SCS logs information about a collector's queue data in the following queue data files:

- checkpointGL_iqfile.dat—lists the collection tables of contents (CTOCs) in the input queue of the collector.
- checkpointGL_oqfile.dat—lists the CTOCs in the output queue of the collector.
- checkpointGL_eqfile.dat—lists the CTOCs in the error queue of the collector. CTOCs move to the error queue of a collector when the collector has made an unsuccessful attempt to send out or take in information. The CTOC remains in the error queue until the collector makes the maximum number of retries. After the collector reaches the maximum number of retries, it sends an error notification and removes the CTOC from all its queues.
- checkpointGL_dqfile.dat—lists the CTOCs in the deferred queue of the collector. CTOCs exist in the deferred queue when links to the collector are turned off. For more information about turning off links, see "Scheduling Collections" on page 4-14 and "wcollect" on page B-8.

Note: The .dat files for the queues exist so that if a collector shuts down, SCS can recover and continue processing the collected data where it left off.

You can view these files using the **wcstat –q** command.

These files are 4 bytes in size when empty and grow in size when the corresponding queue contains data. You can check the size of these files to troubleshoot queues. For example, to verify that the input queue contains data, you can verify that the queue data file checkpointGL_iqfile.dat is greater than 4 bytes in size.

Collector Log File

The mcollect.log file contains all the activity of a collector as data flows through it. You control the amount of information that is logged in this

file by setting a value with the **wcollect –d** option. By default, only fatal errors are logged. You specify the maximum size of this file using the **wcollect –g** option. By default, the maximum size is 1 megabyte (MB). For more information, see "wcollect" on page B-8.

Watch the logging activity as it occurs by running the **tail** command on the collector:

```
tail -f mcollect.log
```

After you run this command, the log file displays a scrolling list of logging activity. If the log file display freezes in the middle of a collection, the file has reached its maximum size. Repeat the **tail** command to resume watching the log activity.

As you watch the mcollect.log activity, be alert for the strings WARNING, ERROR, and exception, which indicate error conditions.

Inventory Data Handler Log File

The inventory data handler creates a log file similar to the log file created by each collector. The file name is the same, mcollect.log, and you configure and monitor this log file using the same commands that you use for the collector log file.

As you monitor this log file, watch for the string IR, which indicates an inventory data handler message. For more information on configuring and monitoring this log file, see "Collector Log File" on page E-9.

If the inventory data handler is having trouble connecting to the configuration repository, check mcollect.log for messages. The inventory data handler writes messages to this log file that indicate which RIM object it is trying to use to connect to the configuration repository.

Collection Manager Log File

The collection manager is installed on the same system as your Tivoli Management Region (TMR) server. The collection manager log file is $TMPDIR/mcollect_collmgr.log on the TMR server, where $TMPDIR is a temporary directory. This temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows NT: c:\temp

This log file contains the routing information that the collection manager provides to the collectors. You can view this log file using the **tail** command or a text editor. To view this file using the **tail** command, run the command as follows:

```
tail -f mcollect_collmgr.log
```

Tivoli Inventory Status Log File

By default, Tivoli Inventory sends status information about scans to the Tivoli Inventory notice group only. Using the **wsetinvglobal –l** command, you can configure Tivoli Inventory to send status information to a log file.

The default path for this log file is $TMPDIR/inv_scan_*nn*.log where *nn* is the scan ID and $TMPDIR is a temporary directory. The temporary directory is usually one of the following paths:

- For UNIX: /tmp, /usr/tmp, or /var/tmp
- For Windows NT: c:\temp

Note: On UNIX systems, $TMPDIR is the temporary directory returned in the **tmpnam** system call. On Windows NT systems, $TMPDIR is the temporary directory returned in the **GetTempPath** system call. For more information, consult the documentation for your operating system.

This log file provides the profile name, the start time, elapsed time, and the number of targets on which scans are completed or pending.

You can view this log file using the **tail** command or a text editor. To view this file using the **tail** command, run the command as follows:

```
tail -f inv_scan_nn.log
```

CTOC Completed Status Log File

You can configure a collector to maintain a log file that lists the CTOCs that have completed on that collector. For troubleshooting, you can check this log file to see whether a collection completed successfully on a collector. The path for this file is $DBDIR/mcollect/CTOC_log.dat. To view the contents of this file, run the following command:

```
wcstat -q c collector
```

where *collector* is of the form **@ManagedNode:***collector_name* or **@Gateway:***collector_name*.

You use the **wcollect –f** option to specify whether or not the collector writes data to the **CTOC_log.dat** file. By default, collectors write data to the file. To check whether a collector is configured to write data to this file, run the following command:

```
wcollect collector
```

where *collector* is of the form **@ManagedNode:***collector_name* or **@Gateway:***collector_name*.

In the resulting output, check the value of log_completed_ctoc. A value of true specifies that the collector writes data to the log file; false specifies that it does not write data to the file.

Other Log Files

The following log files contain information you might find useful when troubleshooting, such as error messages. These log files are text files that you can view using the **tail** command or a text editor.

- $LCF_DATDIR/lcfd.log—a Tivoli Management Framework message log file for capturing endpoint messages. This file resides on each endpoint. For SCS troubleshooting, you can check this log file for information about exceptions that SCS has created or connectivity problems. The $LCF_DATDIR directory is created when you install Tivoli Management Framework. For more information about the lcfd.log file, see the *Tivoli Management Framework Reference Manual*.
- /tmp/rim_db_log—the log file that contains tracing information for RDBMS Interface Module (RIM) objects. This file resides on the managed node where the RIM object is installed. You use the **wrimtrace** command to enable or disable tracing and to specify the information written to the log file. You can write interobject message (IOM) packet information, relational database management system (RDBMS) errors, or both to the RIM log file.

  By default, the RIM log file is not created. You must enable tracing using the **wrimtrace** command to cause the RIM log file to be created.

  Note: The tracing function is intended for debugging purposes. If enabled for extended periods of time, tracing can

decrease performance and slow the processing of RIM calls considerably.

See the *Tivoli Management Framework Reference Manual* for more information about the **wrimtrace** command and the RIM log file.

Depot Contents

You can check the contents of a collector's depot by checking the depot/*CTOC_ID* directory, where *CTOC_ID* specifies a CTOC by identification number. The data represented by the CTOC is contained in the directory named with the ID for that CTOC. The default path for the depot directory is $DBDIR/mcollect/depot on each collector.

You cannot view the files located in the depot directory, but you can check the directory to verify that it exists and see whether it contains data. You can also verify that the files in this directory contain data by verifying that the file size is greater than 0 bytes.

Troubleshooting Procedures

This section covers resolutions for situations you might encounter when using SCS. For more information about the commands described in this section, see Appendix B, "Commands," and the *Tivoli Management Framework Reference Manual*.

- **The scan completes but the data is not in the database.**

  In an SCS-enabled environment, an inventory profile completes when all targets have been physically scanned, but data has not necessarily been populated in the configuration repository.

  The **wgetscanstat** command reports a scan as finished only when all collected data has also been stored in the configuration repository or a failure occurs. To verify that the data for an endpoint has reached the configuration repository, run the command as follows:

  ```
  wgetscanstat -a -s -f
  ```

  You can also check the Tivoli Inventory notice group or Tivoli Inventory status log file for scan completion information, if the

profile for the scan is configured to send notifications to those sources.

- **The scan seems to take too long to finish.**

  First, use the **wgetscanstat** command to view the progress of the endpoints being scanned. If scans for most of the endpoints have completed, use the **wadminep** command to check whether the endpoints that have not completed are accessible.

  Next, use the **wrimtest** command to check whether the RIM object is able to connect to the database. Run the command as follows:

  ```
  wrimtest -l rim_object_label
  ```

  where *rim_object_label* is the name of RIM object used by the inventory data handler.

  Note: By default, only one RIM object, invdh_1, is used by the inventory data handler. However, it is possible to connect the inventory data handler to more than one RIM object. To properly diagnose the problem, you must perform this test on each RIM object used by the inventory data handler.

  To exit the **wrimtest** utility, enter the **x** command option.

  Also, check the names of the RIM objects to make sure they are named correctly. The RIM object created during installation for use by the inventory data handler must be named **invdh_1**. The RIM object used for queries; the Tivoli Inventory GUI; and the **winvfilter**, **winvrmnode**, and **winvsig** commands must be named **inv_query**.

  You should also make sure all RIM objects are configured correctly. Run the following command for each RIM object to check its configuration:

  ```
  wgetrim rim_object_label
  ```

  where *rim_object_label* is the name of RIM object. For the procedure to check the configuration of RIM objects configured to work with the inventory data handler, see “Creating and Configuring RIM Objects for the Inventory Data Handler” on page 4-11.

- **The profile is not using SCS.**

  To troubleshoot this problem, perform the following actions:

  - Verify that SCS is installed on all gateways and managed nodes in your repeater hierarchy. First, run the **wrpt** command to verify your repeater hierarchy. Next, run the **wlsinst** command to verify that all repeaters have been installed with SCS.
  - Run the **wgetinvdh** command to verify that the inventory data handler exists.
  - Run the **wgetrim** command to verify that one or more RIM objects have been created for use with SCS.
  - Verify that the platform type of the endpoint is supported by SCS.
  - Verify that the platform type of the gateway is supported by SCS. If it is not, Tivoli Inventory will not use SCS to collect from that gateway.

  Note: SCS can currently be installed on supported Solaris, HP-UX, AIX, and Windows NT managed node platforms.

- **The depot on a collector is full.**

  When a depot on a collector is full, the collector adds error information to the CTOC for the attempted collection and moves the CTOC into the error queue. From there the CTOC is passed up to the error queue of each collector in the hierarchy. When the CTOC reaches the inventory data handler, the inventory data handler sends a notification stating that the collection for the affected endpoint failed because the depot of an upstream collector is full.

  To troubleshoot this problem, use the **wcollect** command to check the depot size. If the depot is too small, use the **–z** option of the **wcollect** command to increase the size of the depot on that collector, other collectors on the route, and the inventory data handler.

- **A collector has failed.**

  Check the Tivoli Inventory notice group and Tivoli Inventory status information log file for errors.

  Check to see whether an offlink has been set for the collector. For more information about offlinks see "Scheduling Collections" on page 4-14 and "wcollect" on page B-8.

  Use the **wcollect** command to display the collector's configuration. Check to see if the value for *max_input_retries* is set too low. For example, if this value is set to 1, the collector does not retry data collection after the first failure.

  Check the network connectivity between all the collectors from the endpoint to the inventory data handler. If connectivity is lost between any two collectors in this route, the inventory data handler cannot receive error messages until the network is restored.

  To recover a failed collector, use the **wcollect –h** command to halt the collector if it is still running. Then restart the collector using the **wcollect –s** command.

- **A collection has failed.**

  Perform the same checks that you would for a failed collector, but check all the collectors in the route from the endpoint to the inventory data handler. To determine the route, type the **wrpt** command with no arguments to determine the repeater hierarchy, and then reverse the hierarchy to determine the collector hierarchy.

  Also check connectivity between RIM objects and the configuration repository.

- **The status collector incorrectly indicates that a completed scan is still running.**

  If a non-recoverable error occurs for an endpoint during a scan, and the status collector cannot be notified that a failure occurred, the status collector will think that the scan for that endpoint is still pending and that the scan is not complete.

  First, consider recording the information about the status of all the endpoints. You might not get a final notification, depending on the

notification options you have set. The following command lists the information for the scan:

```
wgetscanstat -a -s -f -p
```

Next, reset the status collector. First, you must kill the status collector process. The status collector process runs on the managed node that hosts the inventory data handler and is named inv_stat_meths. You can determine the name of this managed node using the **wgetinvdh –a** command. To determine the ID for the inv_stat_meths process on UNIX systems, use the **ps** command. To determine the ID for this process on Windows NT systems, use the Task Manager, which you can access by right-clicking the taskbar.

To kill the process on UNIX, use the **kill** command. To kill the process on NT, select the process in Processes tab of the Task Manager, then click the **End Process** button.

Finally, delete any saved data for the scan that did not complete. In the directory $DBDIR/inventory/stat_dir on the managed node that hosts the inventory data handler, delete the file associated with the scan that failed. The filename is inv_stat_*n*.cfg, where *n* is the scan ID.

To ensure data integrity, the next time you perform a scan on any endpoint that was listed as pending in the output from the **wgetscanstat** command, set the **Replace with current results** option for the profile.

- **The Tivoli Inventory notification group contains a message stating that a restore operation could not be performed.**

  When a scan is performed, the status collector maintains a file on disk so that if the status collector process is interrupted (for example, if the machine crashes), status can be restored. If the file on disk is corrupted when the status collector restarts, an error similar to the following is reported to the Tivoli Inventory notice group:

```
 The restore operation could not be performed for file
‘/Tivoli/deploy/1/users/bob/inv_8333/db/fuji.db/inventory/st
at_dir/inv_stat_1.cfg’ scan id ‘1’.  The file contains invalid
data.
```

To delete this error, look in the directory $DBDIR/inventory/stat_dir on the inventory data handler managed node for files named inv_stat_*n*.cfg, where n is an integer. Remove the file that is specified in the error message. In the above example, the file is inv_stat_1.cfg.

- **A pending inventory scan is stuck in the queue of the inventory data handler.**

  First, view information about pending scans and targets using the following command:

  ```
  wgetscanstat -a -p
  ```

  Next, verify that data remains in the inventory data handler queues by checking the queue data files in the $DBDIR/inventory/mc directory on the inventory data handler. If the files are greater than 4 bytes in size, they contain data. For more information about these files, see "Queue Data Files" on page E-9.

  Check the inventory data handler log file for messages that indicate the inventory data handler could not connect to RIM object. You might see the following messages:

  IR: No RIM objects found

  Check to see whether the inv_query and invdh_1 RIM objects exist. If they do not exist, you must create them. For more information, see "Creating and Configuring RIM Objects for the Inventory Data Handler" on page 4-11, and the description of **wcrtrim** in the *Tivoli Management Framework Reference Manual*.

  IR: ERROR: No valid rim objects for InvDataHandler in returned sequence

  No existing RIM object is configured to work with the inventory data handler. You must configure one or more RIM objects for this purpose. For more information, see "Creating and Configuring RIM Objects for the Inventory Data Handler" on page 4-11.

IR: ERROR failure in connecting to database

Use the **wrimtest** command to check the connection to each RIM object that is configured to work with the inventory data handler.

Next, halt and restart the inventory data handler. Halt the inventory data handler using the following command:

```
wcollect -h immediate @InvDataHandler:inv_data_handler
```

Restart the inventory data handler using the following command:

```
wcollect -s @InvDataHandler:inv_data_handler
```

- **A scan fails with the following error: "The specified segment ID (config.dmp) with version (*n*) cannot be removed because it is currently in use. Please try again later."**

  This error usually occurs when a managed node is shared by two TMRs. To resolve this error, follow these steps:

  1. Change the default MDist 2 depot directory on the managed node using the following command:

     ```
     wmdist -s managed_node_name rpt_dir=new_directory
     ```

  2. Re-execute the object dispatcher using the following command:

     ```
     odadmin reexec
     ```

Queries

Problems with queries can result from the following conditions:

- The invdh_1 RIM object fails, causing scan failures.
- The inv_query RIM object fails, causing problems with queries, the Inventory property GUI, or the **winvsig**, **winvfilter**, or **winvrmnode** commands.

To test the RIM objects for scan failures, run the following command.

```
wrimtest -l RIM_object
```

where *RIM_object* is the name of the RIM object, either inv_query or invdh_1.

The following message indicates that the connection succeeded:

```
Opening Regular Session...Session Opened
```

If the connection succeeded, enter `X` to exit **wrimtest**.

If the connection fails, a message similar to the following is displayed:

```
Opening Regular Session...FRWRA0012E The RDBMS server call has
failed.
```

If the connection fails, verify that the RIM object passwords are set correctly. Also check the output generated by **wrimtest** for incorrect settings. Use the **wrimset** command to correct the settings for each RIM object.

If the settings are correct and the problem persists, attempt to connect to the configuration repository using the commands provided with the RDBMS.

Glossary

A

admin role

See authorization role.

administrator

See Tivoli administrator.

authorization role

In a Tivoli environment, a role assigned to Tivoli administrators to enable them to perform their assigned systems management tasks. A role may be granted over the entire Tivoli Management Region or over a specific set of resources, such as those contained in a policy region. Examples of authorization roles include: super, senior, admin, and user.

C

cloning

In the Tivoli environment, an operation that enables a Tivoli administrator to replicate profiles. This capability simplifies the task of creating multiple profiles with similar properties. *See also* prototype profile.

collection

In the Tivoli environment, a container that groups objects on a Tivoli desktop, thus providing the Tivoli administrator with a single view of related resources. Either the Tivoli Management Framework or a Tivoli administrator can create a collection. The contents of a collection are referred to as its members. Examples of collections include the administrator collection, the generic collection, and the monitoring collection; the administrator collection is an example of a collection generated by the Tivoli Management Framework.

collector

In a Tivoli environment, either (a) a repeater site on which Scalable Collection Service is installed or (b) a Scalable Collection Service daemon on a managed node or gateway that stores and then forwards data to other collectors or to the inventory data handler.

configuration repository

In a Tivoli environment, a RIM repository that contains information that is collected or generated and stored by Tivoli Inventory and Tivoli Software Distribution. For example, Tivoli Inventory stores information regarding hardware, software, system configuration, and physical inventory; and Tivoli Software Distribution stores information regarding software package and file package operations.

D

default policy

In the Tivoli environment, a set of resource property values that are assigned to a resource when the resource is created.

downcall

In a Tivoli environment, a method invocation from the TMR server or the gateway "down" to an endpoint. Contrast with *upcall*.

downstream

In a network, pertaining to the direction to which data flows.

In a hierarchical network structure, pertaining to the location of a network entity that is lower in the hierarchy. For example, a client is downstream from a server.

Contrast with *upstream*.

E

endpoint

(1) In a Tivoli environment, a Tivoli client that is the ultimate recipient for any type of Tivoli operation. (2) In a Tivoli environment, a Tivoli

service that runs on multiple operating systems and performs Tivoli operations on those systems, thereby enabling the Tivoli Management Framework to manage the systems as Tivoli clients. (3) In Tivoli IT Director, a managed system or device.

event console

In the Tivoli Enterprise Console product, a graphical user interface (GUI) that enables system administrators to view and respond to dispatched events from the event server. The Tivoli Event Integration Facility does not directly use or affect event consoles.

event server

In the Tivoli Enterprise Console product, a central server that processes events. The event server creates an entry for each incoming event and evaluates the event against a rule base to determine whether it can respond to or modify the event automatically. The event server also updates the event consoles with the current event information. If the primary event server is not available, events can be sent to a secondary event server.

F

full-cycle deployment

The Tivoli approach to creating management-ready applications and to continuously deploy and synchronize applications and system configurations on an enterprise scale.

G

gateway

In a Tivoli environment, software running on a managed node that provides all communication services between a group of endpoints and the rest of the Tivoli environment. This gateway includes the multiplexed distribution (MDist 2) function, enabling it to act as the fanout point for distributions to many endpoints.

H

host

In a Tivoli environment, a computer that serves as a managed node for a profile distribution.

I

IDL

See Interface Definition Language.

Interface Definition Language (IDL)

In CORBA, a declarative language that is used to describe object interfaces, without regard to object implementation.

inventory data handler

In a Scalable Collection Service topology, the Tivoli Inventory object that receives data from an inventory scan and uses one or more connections to send the data to the configuration repository.

J

job

In a Tivoli environment, a resource consisting of a task and its preconfigured execution parameters. Among other things, the execution parameters specify the set of hosts on which the job is to execute.

L

local distribution

In a Tivoli environment, a distribution to target machines in the same Tivoli Management Region as the source machine.

M

managed node

In a Tivoli environment, any managed resource on which the Tivoli Management Framework is installed.

managed resource

In a Tivoli environment, any hardware or software entity (machine, service, system, or facility) that is represented by a database object and an icon on the Tivoli desktop. Managed resources must be a supported resource type in a policy region and are subject to a set of rules. Managed resources include, but are not limited to, managed nodes, task libraries, monitors, profiles, and bulletin boards.

management by subscription

In a Tivoli environment, the concept of managing network resources by creating sets of profiles and distributing the profiles (through profile managers) to physical entities (Tivoli resources), called subscribers.

Management Information Format (MIF)

The Desktop Management Interface (DMI) specification that defines the syntax for describing management information about the hardware and software components that can be installed on a computer system.

MDist 2

A multiplexed distribution service provided by Tivoli Management Framework that enables efficient transfer of data to multiple targets. Administrators can monitor and control a distribution throughout its life cycle. In contrast to MDist 2, another multiplexed distribution service, MDist, lacks these management features.

method

In object-oriented design or programming, the software that implements the behavior specified by an operation.

MIF

See Management Information Format.

N

name registry

In a Tivoli environment, a name service consisting of a two-dimensional table that maps resource names to resource identifiers and corresponding information within a Tivoli Management Region.

network computing

The use of a scalable distributed computing infrastructure that encompasses the key elements of today's networking technologies, such as systems and network management, the Internet and intranets, clients and servers, application programs, databases, transaction processing, and various operating systems and communication protocols.

notice

In a Tivoli environment, a message generated by a systems management operation that contains information about an event or the status of an application. Notices are stored in notice groups.

notice group

In a Tivoli environment, an application- or operation-specific container that stores and displays notices pertaining to specific Tivoli functions. The Tivoli bulletin board is composed of notice groups. A Tivoli administrator can subscribe to one or more notice groups; the administrator's bulletin board contains only the notices that reside in a notice group to which the administrator is subscribed.

O

object path

An absolute or relative path to a Tivoli object, similar to paths in file systems.

object reference

In the Tivoli environment, the object identifier (OID) given to an object during its creation.

object request broker (ORB)

In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.

OID

Object identifier.

ORB

See object request broker.

oserv

The name of the object request broker used by the Tivoli environment. Oserv runs on the Tivoli Management Region server and each Tivoli Management Region client.

P

persist

To be maintained across session boundaries, usually in nonvolatile storage such as a database system or a directory.

persistent

Pertaining to data that is maintained across session boundaries, usually in nonvolatile storage such as a database system or a directory.

policy

In the Tivoli environment, a set of rules that are applied to managed resources. A specific rule in a policy is referred to as a "policy method."

policy region

In a Tivoli environment, a group of managed resources that share one or more common policies. Tivoli administrators use policy regions to model the management and organizational structure of a network computing environment. The administrators can group similar resources, define access to and control the resources, and associate rules for governing the resources. The policy region contains resource types and the list of resources to be managed. A policy region is represented

on the Tivoli desktop by an icon that resembles a capitol building (dome icon). When a Tivoli Management Region (region) is created, a policy region with the same name is also created. In this case, the region has only one policy region. However, in most cases, a Tivoli administrator creates other policy regions and subregions to represent the organization of the region. A region addresses the physical connectivity of resources whereas a policy region addresses the logical organization of resources.

policy subregion

In a Tivoli environment, a policy region created or residing in another policy region. When a policy subregion is created, it initially uses the resource and policy properties of the parent policy region. The Tivoli administrator can later change or customize these properties to reflect the specific needs and differences of the subregion.

populate

In the Tivoli environment, to fill a profile with information that is to be distributed to the subscribing managed resources.

profile

In a Tivoli environment, a container for application-specific information about a particular type of resource. A Tivoli application specifies the template for its profiles; the template includes information about the resources that can be managed by that Tivoli application.

A profile is created in the context of a profile manager; the profile manager links a profile to the Tivoli resource (for example, a managed node) that uses the information contained in the profile. A profile does not have any direct subscribers.

profile manager

In a Tivoli environment, a container for profiles that links the profiles to a set of resources, called "subscribers." Tivoli administrators use profile managers to organize and distribute profiles. A profile manager is created in the context of a policy region and is a managed resource in a policy region. A profile manager can operate in one of the following two modes: Dataless mode, which enables a profile manager to distribute to any Tivoli client (including managed nodes, endpoints, and NetWare managed sites) but not to other profile managers. Database mode, which

enables a profile manager to distribute to any profile manager (dataless or database), all managed nodes, and NetWare managed sites—but not to endpoints.

prototype profile

In the Tivoli environment, a model profile from which a Tivoli administrator can create other profiles, often by cloning the prototype profile.

Q

query

In the Tivoli environment, a combination of statements that are used to search the configuration repository for systems that meet certain criteria.

query library

In the Tivoli environment, a facility that provides a way to create and manage Tivoli queries.

R

RDBMS Interface Module (RIM)

In the Tivoli Management Framework, the module in the distributed object database that contains information about the installation of the relational database management system (RDBMS).

registered name

In the Tivoli environment, the name by which a particular resource is registered with the name registry when it is created.

repeater site

In a Tivoli Management Region, a managed node that is configured with the MDist or MDist 2feature. A repeater site receives a single copy of data and distributes it to the next tier of clients.

resource

See managed resource.

resource type

In the Tivoli environment, one of the properties of a managed resource. Resource types are defined in the default policy for a policy region.

RIM

See RDBMS Interface Module.

RIM host

In a Tivoli Inventory environment, the managed node on which one or more RIM objects is installed. *See also* RIM object.

RIM object

In a Tivoli Inventory environment, the object from which a Tivoli administrator runs the database client software and from which the relational database management system (RDBMS) server is contacted. *See also* RIM host.

role

See authorization role.

S

scalable

Pertaining to the capability of a system to adapt readily to a greater or lesser intensity of use, volume, or demand. For example, a scalable system can efficiently adapt to work with larger or smaller networks performing tasks of varying complexity.

Scalable Collection Service (SCS)

A distributed service that enables efficient, asynchronous collection of large amounts of data across complex networks.

scan

To perform an operation that gathers inventory information.

schema

The set of statements, expressed in a data definition language, that completely describe the structure of a database.

SCS

See Scalable Collection Service.

senior role

See authorization role.

signature

In computer software, the name of an operation and its parameters.

subscriber

In a Tivoli environment, a managed node, a profile manager, an endpoint, or another Tivoli client that is subscribed to a profile manager. Although profiles are distributed to a subscriber, the subscriber may or may not be the final destination of the profile distribution.

subscription

In a Tivoli environment, the process of identifying the subscribers to which profiles will be distributed.

subscription list

In a Tivoli environment, a list that identifies the subscribers to a profile manager. Including a profile manager on a subscription list (in effect, a list within a list) is a way of subscribing several resources simultaneously rather than adding each one individually. In Tivoli Plus modules, a profile manager functions as a subscription list.

super role

See authorization role.

T

target

In the Tivoli environment, a Tivoli client on which a job or other activity is performed.

task

In a Tivoli environment, the definition of an action that must be routinely performed on various managed nodes throughout the network. A task defines the executables to be run when the task is executed, the authorization role required to execute the task, and the user or group name under which the task will execute.

Tivoli administrator

In a Tivoli environment, a system administrator who has been authorized to perform systems management tasks and manage policy regions in one or more networks. Each Tivoli administrator is represented by an icon on the Tivoli desktop.

Tivoli client

In a Tivoli environment, any computer—except the TMR server—on which the Tivoli Management Framework is installed. A TMR client runs oserv and maintains a local object database.

Tivoli desktop

In the Tivoli environment, the desktop that system administrators use to manage their network computing environment.

Tivoli Enterprise Console

A Tivoli product that collects, processes, and automatically initiates corrective actions for system, application, network, and database events; it is the central control point for events from all sources. The Tivoli Enterprise Console product provides a centralized, global view of the network computing environment; it uses distributed event monitors to collect information, a central event server to process information, and distributed event consoles to present information to system administrators.

Tivoli environment

The Tivoli applications, based upon the Tivoli Management Framework, that are installed at a specific customer location and that address network computing management issues across many platforms. In a Tivoli environment, a system administrator can distribute software, manage user configurations, change access privileges, automate operations, monitor resources, and schedule jobs.

Tivoli Inventory

A Tivoli product that enables system administrators to gather hardware and software information for a network computing environment. It scans the managed resources and stores inventory information in the configuration repository.

Tivoli Management Region (TMR)

In the Tivoli environment, a TMR server and the set of clients that it serves. An organization can have more than one TMR. The use of interconnected TMRs is transparent to the user.

Note: A TMR addresses the physical layout of resources whereas a policy region addresses the logical organization of resources.

Tivoli Software Distribution

A Tivoli product that automates software distribution to clients and servers in a network-computing environment. An organization can use this product to install and update applications and software in a coordinated, consistent manner across a network. Tivoli Software Distribution creates software packages (in Version 4) and file packages (in Version 3) and distributes them to predefined subscribers.

TMR

See Tivoli Management Region (TMR).

TMR server

In a Tivoli environment, the server that holds or references the complete set of Tivoli software, including the full object database.

U

upcall

In a Tivoli environment, a method invocation from an endpoint "up" to the gateway. Contrast with *downcall*.

upstream

In a network, pertaining to the direction from which data flows.

In a hierarchical network structure, pertaining to the location of a network entity that is higher in the hierarchy. For example, a server is upstream from a client.

Contrast with *downstream*.

user role

See authorization role.

V

validation policy

In a Tivoli environment, policy that ensures that all resources in a policy region comply with the region's established policy. Validation policy prevents Tivoli administrators from creating or modifying resources that do not conform to the policy of the policy region in which the resources were created.

Index

Symbols

$BINDIR/TME/INVENTORY directory 5-21
$DBDIR directory E-8
$DBDIR/inventory/mc directory E-8
$DBDIR/inventory/stat_dir directory 4-8, B-19
$DBDIR/mcollect directory 4-5, B-9, B-10, E-8, E-11
$DBDIR/mcollect/depot directory 4-6, E-13
$LCF_DATDIR directory E-12
.exe files, scanning for 5-17, B-90, B-104

Numerics

128MB_SUBSCRIPTIONS C-80
256MB_SUBSCRIPTIONS C-80
512MB_SUBSCRIPTIONS C-80

A

absolute path B-4
ACCESSED_TIME, description C-14
accessing Tivoli Inventory Web pages 7-24
ADAPTER_MODEL, description C-26
ADAPTER_TYPE, description C-26
Add MIF Attribute page 7-28
Add MIF Group page 7-24
Add or Delete MIF Attribute page 7-26
Add Scheduled Job dialog box 6-6
admin role A-1
administrator resource types 5-6
AIX_SUBSCRIPTIONS C-80
application label 4-12
applications, combining 1-14
assigning policy D-5
authorization roles
    admin role A-1
    backup role A-2
    Install_client role A-2
    Install_product role A-2
    installation
        application 2-17
        software signatures 3-21
        Tivoli Inventory queries 3-23
    Inventory_end_user role A-1
    Inventory_query role A-1
    Inventory_scan role A-1
    Inventory_view role A-1
    policy
        assigning D-6
        creating D-3
        replacing D-4
    profiles
        cloning 5-25
        creating 5-4
        customizing 5-24
        deleting 5-28
        distributing 6-2
    queries
        creating custom 8-3
        creating default 3-23
        defining subscribers with 8-14
        list of roles A-3
    Query_edit role A-1
    Query_execute role A-1
    Query_view role A-1

authorization roles, *continued*
required for Tivoli Inventory A-2
restore role A-2
RIM_update role 7-23, A-1
RIM_view role 7-23, A-1
running queries 8-11
senior role A-2
software signatures 7-3
super role A-2
user role A-1
viewing inventory information 8-17

B

backup role A-2
basic information
filtering 5-16, B-63, B-90, B-104
scanning 5-16, B-90, B-104
BasicInventoryConfig policy object D-2
bios.ini 7-38
BIOS_DATE, description C-38
BIOS_ID, description C-38
BIOS_ID_BYTES, description C-38
BIOS_MANUFACTURER, description C-39
BIOS_MODEL, description C-39
BIOS_SER_NUM, description C-39
BIOS_STRING, description C-38
bold typeface, use of xviii
bundle
configuring destination 4-8, B-77
configuring interval at which it is sent 4-8, B-19, B-73
configuring maximum number of targets 4-9, B-19, B-74
enabling bundling 4-9, B-35, B-74
viewing information B-34
BUTTONS, description C-24

C

cascading delete 7-21
CDROM_QUERY C-60
CDROM_VIEW C-6
checkpointGL_dqfile.dat E-9
checkpointGL_eqfile.dat E-9
checkpointGL_iqfile.dat E-9
checkpointGL_oqfile.dat E-9
CHECKSUM_CRC32, description C-15
CHECKSUM_MD5, description C-15
CHECKSUM_QUICK, description C-15
checksums, generating 5-16, B-91, B-104
CHIP_FAMILY, description C-40
CHIP_MODEL, description C-40
CHIP_STEPPING, description C-40
choosing
RDBMS server 2-11
RIM host 2-11
Clone Profile dialog box 5-26
cloning profiles 5-25– 5-27
CMPXCHG8B_SUPP, description C-41
collection
completed 1-4
deferred 1-4, 4-14, 4-17, B-12
error conditions 1-4, E-10
failed E-16
scheduling 4-14, 4-17, B-12
using SCS 1-8
collection hierarchy 2-4
collection manager 1-4, 4-5, B-12, E-10
collection table of contents. *See* CTOC
collector
configuring retries 4-7, B-11
configuring the log file 4-6, B-9, B-13, E-9
configuring threads 4-7, 4-18, B-11
definition 1-3

increasing the size of the depot 4-6, B-10, E-15
moving the depot 4-5, B-10
moving the run-time directory 4-5, B-10
resetting B-13
resetting collection routes 2-4, 4-5, B-12
scheduling collections 4-14, B-12
starting 4-5, B-9
stopping 4-5, B-9
troubleshooting E-16
using 4-4
viewing configuration information 4-19, B-9
viewing status information 4-19, B-22
collector hierarchy 1-3, 1-4, 2-4, 4-4, 4-5
Column dialog box 8-9
column name description
ACCESSED_TIME C-14
ADAPTER_MODEL C-26
ADAPTER_TYPE C-26
BIOS_DATE C-38
BIOS_ID C-38
BIOS_ID_BYTES C-38
BIOS_MANUFACTURER C-39
BIOS_MODEL C-39
BIOS_SER_NUM C-39
BIOS_STRING C-38
BUTTONS C-24
CHECKSUM_CRC32 C-15
CHECKSUM_MD5 C-15
CHECKSUM_QUICK C-15
CHIP_FAMILY C-40
CHIP_MODEL C-40
CHIP_STEPPING C-40
CMPXCHG8B_SUPP C-41
COMPUTER_MODEL C-9
COMPUTER_SCANTIME C-9
COMPUTER_SYS_ID C-6, C-7
COND_MOVE_SUPP C-42
CREATED_TIME C-14
CURRENT_ADDR C-26
DEBUG_EXT_PRESENT C-42
DEV_ADDR C-55
DEV_CLASS C-56
DEV_IS_HUB C-56
DEV_NAME C-36
DEV_SUBCLASS C-56
DOMAIN_NAME C-38
DRV_NAME C-45
DRV_VERS C-45
FAST_FLOAT_SAVE C-43
FAST_SYS_CALL C-42
FILE_GROUP C-15
FILE_NAME (installed file) C-14
FILE_NAME (unmatched) C-28
FILE_OWNER C-15
FILE_PATH C-25
FILE_PERMISSIONS C-14
FILE_SIZE (installed file) C-14
FILE_SIZE (unmatched) C-28
FS_ACCESS_POINT C-36
FS_FREE_SIZE_KB C-37
FS_MOUNT_POINT C-36
FS_TOTAL_SIZE_KB C-37
FS_TYPE C-36
FUNCTION_KEYS C-22
HDISK_CYLINDERS C-11
HDISK_HEADS C-12
HDISK_SECTORS C-11
HDISK_SIZE_KB C-12
HEADER_NAME C-13
HEADER_PUBLISHER C-13
HEADER_VERS C-13
HOST_CNTRL C-55
INST_DATE C-26
INST_PCI_ID C-44
IP_ADDR C-19
IP_DOMAIN C-19
IP_GATEWAY C-19

column name description, *continued*
IP_HOSTNAME C-19
IP_PRIMARY_DNS C-19
IP_SECONDARY_DNS C-19
IP_SUBNET C-19
IPX_ADDR C-21
KEYBOARD_TYPE C-22
LINK_SPEED C-21
MACHINECHECK_ARCH C-42
MACHINECHECK_EXCPT C-41
MANUFACTURER (CD-ROM) C-6
MANUFACTURER (floppy drive) C-10
MANUFACTURER (hard drive) C-11
MANUFACTURER (network adapter) C-26
MANUFACTURER (PC processor) C-40
MANUFACTURER (processor) C-47
MANUFACTURER (storage device) C-48
MANUFACTURER (USB) C-56
MAX_PACKET_SIZE C-21
MEDIA_TYPE C-36
MEM_TYPE_RANGE_REG C-41
MMX_TECHNOLOGY C-42
MODEL (CD-ROM) C-6
MODEL (floppy drive) C-10
MODEL (hard drive) C-11
MODEL (storage device) C-48
MODEL_SPECIFIC_REG C-41
MODIFIED_TIME C-14
MOUSE_MODEL C-24
MOUSE_TYPE C-24
NET_NUM C-21
NODE_ADDR C-21
NOW_3_D_ARCH C-43
NUM_OF_PORTS C-56
NW_ACCOUNTING_VERS C-30
NW_CLIB_MAJOR_VERS C-31
NW_CLIB_MINOR_VERS C-31
NW_CLIB_REVISION C-31
NW_DEV_NAME C-29
NW_INET_BRG_SUPP C-31
NW_MAX_CONNS C-29
NW_MAX_CONNS_USED C-30
NW_MAX_VOLS C-29
NW_PRINTSERVR_VERS C-30
NW_QUEING_VERS C-30
NW_REVISION_LEVEL C-30
NW_SER_NUM C-31
NW_SFT_LEVEL C-30
NW_SUB_VERS C-29
NW_TTS_LEVEL C-30
NW_VAP_VERS C-30
NW_VERS C-29
NW_VIRT_CONS C-30
NWVOL_AVAIL_BLKS C-32
NWVOL_AVAIL_SLOTS C-33
NWVOL_BLK_SECTORS C-32
NWVOL_DIR_SLOTS C-32
NWVOL_IS_REMOVABLE C-33
NWVOL_NAME C-32
NWVOL_TOTAL_BLKS C-32
ON_CHIP_APIC C-41
ON_CHIP_FPU C-42
OS_INST_DATE C-35
OS_MAJOR_VERS C-34
OS_MINOR_VERS C-34
OS_NAME C-34
OS_SUB_VERS C-34
OS_TYPE C-34
PACKAGE_ID C-25
PACKAGE_NAME C-25
PACKAGE_VERS C-25
PAGE_ATTR_TABLE C-42
PAGE_GLOBAL_ENABLE C-42
PAGE_SIZE C-7
PAGE_SIZE_EXT C-41
PAGE_SIZE_EXT36 C-42
PARENT_ADDR C-55
PARTITION_TYPE C-36
PATH (installed file) C-14
PATH (unmatched) C-28, C-31

PCI_DEV_NAME C-44
PERM_MAC_ADDR C-26
PHYSICAL_ADDR_EXT C-41
PHYSICAL_FREE_KB C-7
PHYSICAL_SIZE_KB C-36
PHYSICAL_TOTAL_KB C-7
PORT_NAME C-45
PORT_NUM C-55
PRFL_ACTION C-59
PRINTER_IS_LOCAL C-45
PRINTER_LOCATION C-45
PRINTER_MODEL C-45
PRINTER_NAME C-45
PROCESSOR_MODEL (PC processor) C-40
PROCESSOR_MODEL (processor) C-47
PROCESSOR_SPEED (PC processor) C-40
PROCESSOR_SPEED (processor) C-47
PRODUCT C-56
PRODUCT_ID C-56
PUBLISHER C-25
RECORD_ACTION C-58
RECORD_TIME C-6
SER_NUM (CD-ROM) C-6
SER_NUM (hard drive) C-11
SER_NUM (PC processor) C-43
SER_NUM (processor) C-47
SER_NUM (storage device) C-48
SER_NUM (USB) C-55
SER_NUM_ENABLED C-43
SIMD_EXT_SUPP C-43
STORAGE_TYPE (CD-ROM) C-6
STORAGE_TYPE (floppy drive) C-10
STORAGE_TYPE (hard drive) C-11
STORAGE_TYPE (storage device) C-48
SWARE_DESC C-16
SWARE_DESC (CRC32 match) C-49
SWARE_DESC (MD5 match) C-51
SWARE_DESC (Quick match) C-53
SWARE_DESC (software match) C-23
SWARE_NAME C-16
SWARE_NAME (CRC32 match) C-49
SWARE_NAME (MD5 match) C-51
SWARE_NAME (Quick match) C-53
SWARE_NAME (software match) C-23
SWARE_SIZE C-16
SWARE_SIZE (CRC32 match) C-50
SWARE_SIZE (MD5 match) C-51
SWARE_SIZE (Quick match) C-53
SWARE_SIZE (software match) C-23
SWARE_VERS C-16
SWARE_VERS (CRC32 match) C-49
SWARE_VERS (MD5 match) C-51
SWARE_VERS (Quick match) C-53
SWARE_VERS (software match) C-23
TIME_STAMP_COUNTER C-41
TME_OBJECT_ID C-6
TME_OBJECT_LABEL C-6
TOTAL_PAGES C-7
USB_VERS C-55
USER_NAME C-38
VENDOR_ID C-56
VID_BIOS_RELDATE C-57
VID_CARD_BIOS C-57
VID_CARD_MODEL C-57
VID_CHIP_TYPE C-57
VID_COLORS C-58
VID_DAC_TYPE C-57
VID_HORIZNTL_RES C-58
VID_MEM C-57
VID_VERTICAL_RES C-58
VIRT_FREE_KB C-8
VIRT_MODE_EXT C-41
VIRT_TOTAL_KB C-7
WORKGROUP_NAME C-38

columns
for DB2 and Informix RDBMSs 7-14, 7-16
naming 7-14
combining applications 1-14
commands
dmiscan E-4
for SCS B-6
for Tivoli Inventory B-4
idlcall 4-13
odadmin 2-4, 4-16, E-8, E-19
overview B-1
syntax B-2
tail E-10, E-11, E-12
wadminep E-2, E-14
wbkupdb 2-17
wcollect 2-4, 4-4– 4-9, 4-16– 4-19, B-8, E-10– E-19
wcomprules 2-36
wcprb 2-36
wcrtadmin 7-12
wcrtinvcb 4-11, B-16
wcrtinvdh 4-8, 4-10, 4-22, B-18, B-34– B-35, B-73
wcrtpol D-3
wcrtprf 5-27
wcrtquery 7-22
wcrtrb 2-36
wcrtrim 2-28, 4-12
wcstat 4-19, B-13, B-22, E-9
wdel 4-10, 4-11, 5-29
wdistinv 6-7, B-25
wepscan 6-8, B-31, E-5
wgetinvdh 4-20, B-34, E-15
wgetinvglobal 4-21, 4-22, B-37
wgetinvpcfiles B-41
wgetinvpchw B-43
wgetinvpcsw B-46
wgetinvunixfiles B-51
wgetinvunixhw B-53
wgetinvunixsw B-56
wgetpolm D-4
wgetrim 2-12, E-15
wgetscanstat 4-20, 4-22, B-61, E-13– E-14
wimprbclass 2-36
winstall 2-29
winvfilter B-63
winvrmnode 2-35, 7-21, B-65
winvsig 3-21, 7-3, 7-4, B-67
wloadrb 2-36
wlookup 4-13
wlsinst E-15
wlspolm D-4
wmdist 6-6
wpatch 2-10
wputpolm D-5
wqueryinv B-71
wrimset E-20
wrimtest E-14, E-19
wrimtrace E-12
wrpt 4-4, 4-5, B-12, E-15, E-16
wruninvquery 8-16
wrunquery 8-11, 8-13, 8-16
wsetadmin 7-13
wsetinvdh 4-8, 4-21– 4-22, B-19, B-34– B-35, B-73
wsetinvglobal 4-21– 4-22, B-18, B-19, B-21, B-35, B-38, B-73– B-74, B-76, E-11
wsetinvpcfiles B-81
wsetinvpchw B-86
wsetinvpcsw B-90
wsetinvunixfiles B-95
wsetinvunixhw B-100
wsetinvunixsw B-104
wsetpr D-6
wsetrim 2-12, 4-14
wsetrimpw 2-12, 2-29, 3-4, 3-8, 3-19
wstartesvr 2-37
wstopesvr 2-36
wuninst 2-33
components of Tivoli Inventory 1-2

COMPUTER table C-83
COMPUTER_MEM_QUERY C-61
COMPUTER_MEM_VIEW C-7
COMPUTER_MODEL, description C-9
COMPUTER_SCANTIME, description C-9
COMPUTER_SYS_ID, description C-6, C-7
COMPUTER_SYS_MEM table C-84
COMPUTER_VIEW C-9
COND_MOVE_SUPP, description C-42
config.dmp B-32
configuration files
  distributing 5-9, B-77
  policy D-13
  writing contents to a log file B-32
configuration repository
  adding tables 7-20
  adding views 7-34
  cascading delete 7-21
  configuring scans to send data 5-12, B-90, B-104
  creating for an RDBMS
    DB2 2-12, 3-3– 3-6
    Informix 2-14, 3-8– 3-9
    MS SQL Server 3-9– 3-13
    Oracle 3-13– 3-17
    overview 3-1
    Sybase 3-17– 3-20
  customizing 7-14– 7-36
  default queries C-60– C-82
  definition 1-3
  deleting data for one endpoint B-65
  editing 7-14– 7-36
  historical subscription queries C-81– C-82
  historical views C-58– C-60
  modifying filter list 5-16, B-63
  modifying software signatures 7-2, B-67
  overview C-83, C-105
  populating 6-9
  schemas 3-4, 3-8, 3-11, 3-15, 3-19
  subscription queries C-80– C-82
  tables
    provided with Tivoli Inventory C-83– C-105
  troubleshooting E-1
  viewing inventory data 4-21
  views C-5– C-60
conventions used xviii
copying profiles 5-25
CRC. *See* checksums
Create Administrator dialog box 7-9
Create Profile dialog box 5-6
Create Query dialog box 8-3, 8-5
CREATED_TIME, description C-14
creating
  configuration repositories 3-1– 3-25
  history tables for custom tables 7-19
  history tables for DB2 3-5
  history tables for Informix 3-8
  history tables for MS SQL Server 3-12
  history tables for Oracle 3-16
  history tables for Sybase 3-20
  inventory callback object B-16
  inventory data handler B-18
  profiles 5-3– 5-8
  queries 8-2– 8-10
  RIM objects 4-11
  software signatures 7-2– 7-5
  User Data Form 7-23– 7-30
  useradd.mif file 7-23– 7-30
CTOC
  description 1-4
  role in data collection 1-9, 1-10, 4-3, 4-4
  viewing information about 4-19, B-22
CTOC_log.dat E-12

CURRENT_ADDR, description C-26
custom MIF files
naming 7-15, 7-36
policy D-21
using 7-36– 7-39
customizing
configuration repository 7-14– 7-36
hardware scans 5-18– 5-23, B-86, B-100
profiles 5-8– 5-25
queries 8-2– 8-10
software scans 5-10– 5-18, B-90, B-104

D

data units 4-6, 4-17, B-10
dataless endpoint mode 5-3
DB2
configuration 2-12
creating a configuration repository 3-2– 3-6
creating an instance 2-13
RIM host 2-12
DEBUG 3 log file E-7
DEBUG_EXT_PRESENT, description C-42
Delete Profiles dialog box 5-28
deleting
profiles 5-27– 5-29
scan data for one endpoint B-65
software signatures 7-2– 7-5
depot
checking contents E-13
default location 4-6, B-10
definition 1-4
increasing the size 4-6, B-10
moving 4-5, B-10
troubleshooting E-15
DEV_ADDR, description C-55
DEV_CLASS, description C-56
DEV_IS_HUB, description C-56
DEV_NAME, description C-36
DEV_SUBCLASS, description C-56
dialog box
Add Scheduled Job 6-6
Clone Profile 5-26
Column 8-9
Create Administrator 7-9
Create Profile 5-6
Create Query 8-3, 8-5
Delete Profiles 5-28
DMI Scanner Configuration 5-20
Export Query Results 8-12
File Browser 2-7
Install Patch 2-6, 2-8, 2-9
Inventory Profile 6-4
Patch Install 2-9, 2-10
Run Query 8-12
Set Login Names 7-10
Set TMR Roles 7-10
Software Scan Configuration 5-14
Timeout Settings 6-5
Tivoli Hardware Scanner 5-19, 5-23
directories, rules for specifying B-82, B-96
directory
$BINDIR/TME/INVENTORY 5-21
$DBDIR E-8
$DBDIR/inventory/mc E-8
$DBDIR/inventory/stat_dir 4-8, B-19
$DBDIR/mcollect 4-5, B-9, B-10, E-8, E-11
$DBDIR/mcollect/depot 4-6, E-13
$LCF_DATDIR E-12
distributing profiles
CLI information B-25
overview 6-1
procedure 6-3– 6-8

DMI
customizing scans 5-19
dmi_*RDBMS_type* _schema.sql script 5-21, E-4
error messages E-2
naming tables and columns E-4
scanner 5-19
troubleshooting E-1– E-4
DMI Scanner Configuration dialog box 5-20
dmi_*RDBMS_type* _schema.sql 5-21, E-4
dmiscan E-4
DOMAIN_NAME, description C-38
DRV_NAME, description C-45
DRV_VERS, description C-45

E

editing
configuration repository 7-14– 7-36
software signatures 7-2– 7-5
User Data Form 7-23– 7-30
useradd.mif file 7-23– 7-30
endpoint-initiated scans
CLI information B-31
description 6-8
prerequisite tasks E-6
setting up environment B-31
troubleshooting E-4– E-6
writing scan data to a log file B-32
endpoints
definition 1-6
deleting scan data B-65
error conditions 1-4, E-2, E-10
EXE files, scanning for 5-17, B-90, B-104
Export Query Results dialog box 8-12

F

FAST_FLOAT_SAVE, description C-43
FAST_SYS_CALL, description C-42
features of Tivoli Inventory 1-10
File Browser dialog box 2-7
file extensions, policy D-24, D-36
file names, rules for specifying B-83, B-96
FILE_DESC table C-85
FILE_FILTER table C-85
FILE_GROUP, description C-15
FILE_NAME (installed file), description C-14
FILE_NAME (unmatched), description C-28
FILE_OWNER, description C-15
FILE_PATH table C-86
FILE_PATH, description C-25
FILE_PERMISSIONS, description C-14
FILE_SIZE (installed file), description C-14
FILE_SIZE (unmatched), description C-28
filter
scans for .exe files only 5-17, B-90, B-104
scans for basic information 5-16, B-63, B-90, B-104
FLPY_DRV_QUERY C-61
FLPY_DRV_VIEW C-10
foreign keys 7-15, 7-21
FS_ACCESS_POINT, description C-36
FS_FREE_SIZE_KB, description C-37
FS_MOUNT_POINT, description C-36
FS_TOTAL_SIZE_KB, description C-37
FS_TYPE, description C-36
full-cycle deployment 1-14
FUNCTION_KEYS, description C-22

G

gateway, definition 1-6
GUI
enabling log file E-6
enabling UNIX connections E-7
system requirements E-6
troubleshooting E-6– E-8

H

H_AIX_SUBSCRIPTIONS C-82
H_CDROM_QUERY C-79
H_CDROM_VIEW C-59
H_COMPUTER table C-104
H_COMPUTER_MEM_QUERY C-79
H_COMPUTER_QUERY C-79
H_COMPUTER_SYS_MEM table C-104
H_COMPUTER_VIEW C-59
H_FLPY_DRV_QUERY C-79
H_FLPY_DRV_VIEW C-59
H_HDISK_QUERY C-79
H_HDISK_VIEW C-59
H_HEADER_INFO_QUERY C-79
H_HEADER_VIEW C-59
H_HPUX_SUBSCRIPTIONS C-82
H_INST_FILE_QUERY C-79
H_INST_FILE_VIEW C-59
H_INST_HEADER_INFO table C-104
H_INST_MOUSE table C-104
H_INST_NATIVE_SWARE table C-104
H_INST_PARTITION table C-104
H_INST_PRINTER table C-104
H_INST_PROCESSOR table C-104
H_INST_SWARE_VIEW C-59
H_INST_USB_DEV table C-104
H_INST_VID_CARD table C-104
h_inv_db2_schema.log 3-5
h_inv_db2_schema.sql script 3-5
h_inv_infx_schema.log 3-8
h_inv_infx_schema.sql script 3-8
h_inv_ms_schema.log 3-12
h_inv_ms_schema.sql script 3-12
h_inv_ora_schema.log 3-16
h_inv_ora_schema.sql script 3-16
h_inv_syb_schema.log 3-20
h_inv_syb_schema.sql script 3-20
h_inventory_query.sh script 3-22
H_INVENTORY_SWARE query C-79
H_IP_ADDR table C-104
H_IP_ADDR_QUERY C-79
H_IP_ADDR_VIEW C-59
H_IPX_ADDR table C-104
H_IPX_ADDR_QUERY C-79
H_IPX_ADDR_VIEW C-59
H_MATCHED_SWARE table C-104
H_MEM_VIEW C-59
H_MOUSE_QUERY C-79
H_MOUSE_VIEW C-59
H_NATIV_SWARE_QUERY C-79
H_NATIV_VIEW C-59
H_NET_ADAPTER table C-104
H_NET_CARD_QUERY C-79
H_NET_CARD_VIEW C-59
H_NETWARE_SUBSCRIPTIONS C-82
H_NOSIG_FILES_QUERY C-79
H_NOSIG_FILES_VIEW C-59
H_NW_SERVER table C-104
H_NW_SERVER_QUERY C-79
H_NW_SERVER_VIEW C-59
H_NW_VOLS table C-104
H_NW_VOLS_QUERY C-79
H_NW_VOLS_VIEW C-59
H_OS_QUERY C-79
H_OS_VIEW C-59
H_OS2_SUBSCRIPTIONS C-82
H_PARTITION_QUERY C-79

H_PARTITION_VIEW C-59
H_PC_BIOS_QUERY C-79
H_PC_BIOS_VIEW C-59
H_PC_SYS_PARAMS table C-104
H_PCI_DEV table C-104
H_PCI_DEV_QUERY C-79
H_PCI_DEV_VIEW C-59
H_PCPROCESSOR_QUERY C-79
H_PCPROCESSOR_VIEW C-59
H_PRINTER_QUERY C-79
H_PRINTER_VIEW C-59
H_PROCESSOR_QUERY C-79
H_PROCESSOR_VIEW C-59
H_Solaris_SUBSCRIPTIONS C-82
H_STORAGE_DEV table C-104
H_STORAGE_DEV_QUERY C-79
H_STORAGE_DEV_VIEW C-59
h_subscription_query.sh script 3-22
H_TAPEDRV_QUERY C-79
H_TAPEDRV_VIEW C-60
H_UNIX_SYS_PARAMS table C-105
H_UNIX_SYS_QUERY C-79
H_UNIX_SYS_VIEW C-60
H_UNMATCHED_FILES table C-105
H_USB_DEV_QUERY C-80
H_USB_DEV_VIEW C-60
H_VID_CARD_QUERY C-80
H_VID_CARD_VIEW C-60
H_WIN_2000_SUBSCRIPTIONS C-82
H_WIN_98_SUBSCRIPTIONS C-82
H_WIN_ALL_SUBSCRIPTIONS C-82
H_WIN_ME_SUBSCRIPTIONS C-82
H_WIN_NT_SUBSCRIPTIONS C-82
Hardware window 5-18, 5-22
HDISK table C-86
HDISK_CYLINDERS, description C-11
HDISK_HEADS, description C-12
HDISK_QUERY C-62
HDISK_SECTORS, description C-11
HDISK_SIZE_KB, description C-12
HDISK_VIEW C-11
header information, scanning 5-16, B-90
HEADER_INFO table C-87
HEADER_INFO_QUERY C-62
HEADER_INFO_VIEW C-13
HEADER_NAME, description C-13
HEADER_PUBLISHER, description C-13
HEADER_VERS, description C-13
hierarchy
  collector 1-3, 1-4, 2-4, 4-4, 4-5
  repeater 1-3, 2-4, 4-4
history tables
  creating for custom tables 7-19
  creating for DB2 3-5
  creating for Informix 3-8
  creating for MS SQL Server 3-12
  creating for Oracle 3-16
  creating for Sybase 3-20
  using C-3
history tracking
  enabling C-2
  overview C-2
  using history tables C-3
HOST_CNTRL, description C-55
HPUX_SUBSCRIPTIONS C-80

I

icons, use of xviii
idlcall 4-13
Informix
  configuration 2-14
  creating a configuration repository 3-7– 3-8
  RIM host 2-15
INST_DATE, description C-26
INST_FILE_QUERY C-63

INST_FILE_VIEW C-14
INST_HEADER_INFO table C-87
INST_MOUSE table C-88
INST_NATIV_SWARE table C-88
INST_PARTITION table C-89
INST_PCI_ID, description C-44
INST_PRINTER table C-89
INST_PROCESSOR table C-90
INST_SWARE_VIEW C-16
INST_USB_DEV table C-90
INST_VID_CARD table C-91
Install Patch dialog box 2-6, 2-8, 2-9
Install_client role A-2
Install_product role A-2
installing
  choosing between Tivoli Inventory Server and Gateway 2-21
  queries 3-22
  SCS from the command line 2-10
  SCS from the desktop 2-6
  software signatures 3-21
  subscription queries 3-22
  Tivoli Inventory
    overview 2-1
    with Software Installation Service 2-3
  Tivoli Inventory Gateway 2-31
  Tivoli Inventory Server
    on managed nodes 2-21– 2-31
    on TMR server 2-21– 2-31
integrating applications 1-14
interconnected TMRs 2-12, 3-23, 4-22, 8-20
interfaces file 2-16
inv_40 2-28
inv_db2_admin.sql script 3-3
inv_db2_schema.sql script 3-4
inv_infx_schema.sql script 3-8
inv_ms_sql_admin.sql script 3-10
inv_ms_sql_schema.sql script 3-11
inv_ora_admin.sql script 3-13
inv_ora_schema.sql script 3-15
inv_query 2-28
INV_SA.DAT B-32
inv_scan_*nn*.log E-11
inv_syb_admin.sql script 3-17
inv_syb_schema.sql script 3-19
invdh_1 2-28– 2-29, 4-11
inventory callback object
  choosing host 2-23
  creating B-16
  definition 1-5
  moving 4-10
inventory data handler
  choosing host 2-23
  configuring 4-8, B-18, B-73
  configuring output threads 4-9
  configuring retries 4-9, B-20, B-74
  configuring status information 4-8, 4-21, B-19, B-73
  configuring timeout between retries 4-9, B-20, B-75
  creating B-18
  definition 1-4
  moving 4-10
  using 4-7
  viewing configuration information 4-20, B-34
Inventory Profile dialog box 6-4
inventory profiles. *See* profiles
inventory RIM object 2-28
Inventory_end_user role A-1
INVENTORY_HWARE C-63
Inventory_query role A-1
inventory_query.sh script 3-22
Inventory_scan role A-1
INVENTORY_SWARE C-64
Inventory_view role A-1

InventoryConfig
adding to a policy region 2-27, 5-4–5-6
setting defaults D-1
INVENTORYDATA C-17
IP_ADDR table C-92
IP_ADDR, description C-19
IP_ADDR_QUERY C-65
IP_ADDR_VIEW C-19
IP_DOMAIN, description C-19
IP_GATEWAY, description C-19
IP_HOSTNAME, description C-19
IP_PRIMARY_DNS, description C-19
IP_SECONDARY_DNS, description C-19
IP_SUBNET, description C-19
IPX_ADDR table C-92
IPX_ADDR, description C-21
IPX_ADDR_QUERY C-65
IPX_ADDR_VIEW C-21
italic typeface, use of xviii

K

KEYBOARD_QUERY C-66
KEYBOARD_TYPE, description C-22
KEYBOARD_VIEW C-22

L

lcfd.log E-12
LINK_SPEED, description C-21

M

MACHINECHECK_ARCH, description C-42
MACHINECHECK_EXCPT, description C-41
managed resources, setting defaults D-1
management by subscription 1-5
Management Information Format files. *See* MIF files
MANUFACTURER (CD-ROM), description C-6
MANUFACTURER (floppy drive), description C-10
MANUFACTURER (hard drive), description C-11
MANUFACTURER (network adapter), description C-26
MANUFACTURER (PC processor), description C-40
MANUFACTURER (processor), description C-47
MANUFACTURER (storage device), description C-48
MANUFACTURER (USB), description C-56
MATCH_CRC32_QUERY C-66
MATCH_MD5_QUERY C-67
MATCH_QUICK_QUERY C-67
MATCH_SWARE_VIEW C-23
MATCHED_SWARE table C-93
MAX_PACKET_SIZE, description C-21
mcollect.log 4-6, B-9, E-9, E-10
MD5. *See* checksums
MDist 2 1-3, 2-4, 6-3
MEDIA_TYPE, description C-36
MEM_TYPE_RANGE_REG, description C-41

MIF files
custom
naming 7-36
policy D-21
using with Tivoli Inventory 5-23– 5-24, 7-36– 7-39, B-81, B-95
definition 1-6
example 7-37
generated by Tivoli Inventory 7-37
MIF Groups and Attributes page 7-30
MMX_TECHNOLOGY, description C-42
mobile endpoints
hiding distributions B-28, B-78
mandatory distributions B-27
reminder messages B-27
support 1-12, B-29
MODEL (CD-ROM), description C-6
MODEL (floppy drive), description C-10
MODEL (hard drive), description C-11
MODEL (storage device), description C-48
MODEL_SPECIFIC_REG, description C-41
MODIFIED_TIME, description C-14
monospace typeface, use of xviii
MOUSE table C-93
MOUSE_MODEL, description C-24
MOUSE_QUERY C-68
MOUSE_TYPE, description C-24
MOUSE_VIEW C-24
mrmbios 7-38
MS SQL Server
configuration 2-15
creating a configuration repository 3-9– 3-13
RIM host 2-15
multiplexed distribution. *See* MDist 2

N

name registry 4-22, B-3
naming
custom MIF files 7-15, 7-36
profiles 5-3
queries 3-23, 8-5
query libraries 3-23
tables and columns 7-14
tables and columns for DB2 and Informix 7-14, 7-16
NATIV_SWARE table C-94
NATIV_SWARE_QUERY C-68
NATIV_SWARE_VIEW C-25
NET_ADAPTER table C-94
NET_CARD_QUERY C-69
NET_CARD_VIEW C-26
NET_NUM, description C-21
NETWARE_SUBSCRIPTIONS C-80
nobody account 4-6, B-10
NODE_ADDR, description C-21
NOSIG_FILES_QUERY C-69
NOSIG_FILES_VIEW C-28
notice group 6-2
notification
See status information
NOW_3_D_ARCH, description C-43
NUM_OF_PORTS, description C-56
NW_ACCOUNTING_VERS, description C-30
NW_CLIB_MAJOR_VERS, description C-31
NW_CLIB_MINOR_VERS, description C-31
NW_CLIB_REVISION, description C-31
NW_DEV_NAME, description C-29
NW_INET_BRG_SUPP, description C-31
NW_MAX_CONNS, description C-29
NW_MAX_CONNS_USED, description C-30

NW_MAX_VOLS, description C-29
NW_PRINTSERVR_VERS, description C-30
NW_QUEING_VERS, description C-30
NW_REVISION_LEVEL, description C-30
NW_SER_NUM, description C-31
NW_SERVER table C-95
NW_SERVER_QUERY C-69
NW_SERVER_VIEW C-29
NW_SFT_LEVEL, description C-30
NW_SUB_VERS, description C-29
NW_TTS_LEVEL, description C-30
NW_VAP_VERS, description C-30
NW_VERS, description C-29
NW_VIRT_CONS, description C-30
NW_VOLS_QUERY C-71
NW_VOLS_VIEW C-32
NWVOL_AVAIL_BLKS, description C-32
NWVOL_AVAIL_SLOTS, description C-33
NWVOL_BLK_SECTORS, description C-32
NWVOL_DIR_SLOTS, description C-32
NWVOL_IS_REMOVABLE, description C-33
NWVOL_NAME, description C-32
NWVOL_TOTAL_BLKS, description C-32

O

object dispatcher number
  definition 8-21
  specifying offlinks 4-16, B-12
object ID 8-20
object number 8-21
object paths B-4
object references B-2
odadmin 2-4, 4-16, E-8, E-19
offlink 4-14, 4-17, B-12, E-16
ON_CHIP_APIC, description C-41
ON_CHIP_FPU, description C-42
opening Tivoli Inventory Web pages 7-24
operational data tables C-2
Oracle
  configuration 2-15
  creating a configuration repository 3-13– 3-17
  RIM host 2-16
OS_INST_DATE, description C-35
OS_MAJOR_VERS, description C-34
OS_MINOR_VERS, description C-34
OS_NAME, description C-34
OS_QUERY C-71
OS_SUB_VERS, description C-34
OS_TYPE, description C-34
OS_VIEW C-34
OS2_SUBSCRIPTIONS C-80
oserv 4-6

P

PACKAGE_ID, description C-25
PACKAGE_NAME, description C-25
PACKAGE_VERS, description C-25
page
  Add MIF Attribute 7-28
  Add MIF Group 7-24
  Add or Delete MIF Attribute 7-26
  MIF Groups and Attributes 7-30
  Results 7-26, 7-29
  User Data Form 7-30, 7-33
  User Data Template 7-24
  UserLink for Tivoli 7-32
  UserLink for Tivoli Inventory 7-33

PAGE_ATTR_TABLE, description C-42
PAGE_GLOBAL_ENABLE, description C-42
PAGE_SIZE, description C-7
PAGE_SIZE_EXT, description C-41
PAGE_SIZE_EXT36, description C-42
PARENT_ADDR, description C-55
PARTITION_QUERY C-72
PARTITION_TYPE, description C-36
PARTITION_VIEW C-36
passwords, changing for RIM objects 2-28
Patch Install dialog box 2-9, 2-10
path
  absolute B-4
  relative B-4
PATH (installed file), description C-14
PATH (unmatched), description C-28, C-31
PC hardware scans 5-18, B-86
PC_BIOS_QUERY C-72
PC_BIOS_VIEW C-38
PC_PROCESSOR_QUERY C-73
PC_PROCESSOR_VIEW C-40
PC_SYS_PARAMS table C-96
PCI_DEV table C-97
PCI_DEV_NAME, description C-44
PCI_DEV_QUERY C-74
PCI_DEV_VIEW C-44
PERM_MAC_ADDR, description C-26
PHYSICAL_ADDR_EXT, description C-41
PHYSICAL_FREE_KB, description C-7
PHYSICAL_SIZE_KB, description C-36
PHYSICAL_TOTAL_KB, description C-7
policy
  assigning to a policy region D-5
  creating D-3
  default D-2
  ic_def_global_action D-13
  ic_def_global_ep_timeout D-14
  ic_def_global_logfile_host D-15
  ic_def_global_logfile_path D-16
  ic_def_global_notice_interval D-17
  ic_def_global_notice_location D-18
  ic_def_global_notice_type D-19
  ic_def_global_update_replace D-20
  ic_def_pc_custom_mifs D-21
  ic_def_pc_custom_script D-22
  ic_def_pc_exclude_dirs D-23
  ic_def_pc_extensions D-24
  ic_def_pc_hw_gran D-25
  ic_def_pc_hw_outfile D-26
  ic_def_pc_hw_scan D-27
  ic_def_pc_include_dirs D-28
  ic_def_pc_sw_crc D-29
  ic_def_pc_sw_flags D-30
  ic_def_pc_sw_outfile D-31
  ic_def_pc_sw_scan D-32
  ic_def_unix_custom_mifs D-33
  ic_def_unix_custom_script D-34
  ic_def_unix_exclude_dirs D-35
  ic_def_unix_extensions D-36
  ic_def_unix_hw_gran D-37
  ic_def_unix_hw_outfile D-38
  ic_def_unix_hw_scan D-39
  ic_def_unix_include_dirs D-40
  ic_def_unix_sw_crc D-41
  ic_def_unix_sw_flags D-42
  ic_def_unix_sw_outfile D-43
  ic_def_unix_sw_scan D-44
  overview D-1
  replacing D-4
  setting a default method D-6
  Tivoli Inventory D-1– D-44
policy region icons xviii
populating the configuration repository 6-9
PORT_NAME, description C-45
PORT_NUM, description C-55
prerequisite documents xv
PRFL_ACTION, description C-59

primary keys 7-15, 7-36
PRINTER table C-98
PRINTER_IS_LOCAL, description C-45
PRINTER_LOCATION, description C-45
PRINTER_MODEL, description C-45
PRINTER_NAME, description C-45
PRINTER_QUERY C-75
PRINTER_VIEW C-45
PROCESSOR table C-98
PROCESSOR_MODEL (PC processor), description C-40
PROCESSOR_MODEL (processor), description C-47
PROCESSOR_QUERY C-75
PROCESSOR_SPEED (PC processor), description C-40
PROCESSOR_SPEED (processor), description C-47
PROCESSOR_VIEW C-47
PRODUCT, description C-56
PRODUCT_ID, description C-56
Profile Manager window 5-4, 5-7
profile managers
  definition 1-5
  using 5-3
profiles
  cloning 5-25– 5-27
  collecting custom MIF files 5-23, B-81, B-95
  copying 5-25
  creating 5-3– 5-8
  customizing 1-11, 5-8– 5-25
  definition 1-5
  deleting 5-27– 5-29
  distributing 6-3– 6-8, B-25
  hiding distribution to roaming endpoints B-28, B-78
  mandatory distributions to roaming endpoints B-27
  naming 5-3
organizing 5-3
overview of distribution 6-1
recovering 5-28
reminder messages to roaming endpoints B-27
renaming 5-29
sending data to the configuration repository 5-12, B-90, B-104
setting custom scripts to run 5-23, B-81, B-95
setting data options 5-9, B-79
setting defaults for new D-1
setting directories to scan 5-17, B-81, B-95
setting distribution deadline 6-5, B-26
setting distribution options 5-9, B-77
setting files to scan 5-17, B-81, B-95
setting global properties 5-9, B-76
setting hardware scan options 5-18– 5-23, B-86, B-100
setting notification interval 6-5, B-26
setting priority 6-3, 6-4, B-25
setting software scan options 5-10– 5-18, B-90, B-104
setting status options 4-21– 4-22, B-76
setting timeout for endpoint response 6-6, B-26
setting timeout for profile distribution 6-5, B-26
setting timeout for scan attempt B-77
support for roaming endpoints B-29
troubleshooting E-15
using 5-2
viewing configuration information B-41, B-51
viewing global properties 4-21, B-37
viewing hardware scan options B-43, B-53

profiles, *continued*
viewing information about 4-21, B-37
viewing software scan options B-46, B-56
wake-on-LAN support B-29, B-79
PUBLISHER, description C-25

Q

queries
authorization roles A-3
creating 8-2– 8-10
creating libraries 8-4
customizing 8-6
default for Tivoli Inventory
creating 3-22
list of C-60– C-82
defining subscribers 8-14
defining targets 8-14
example 8-3– 8-10
executing 8-10– 8-13
executing from the CLI B-71
executing from the icon menu 8-18
in interconnected TMRs 8-20
overview 8-1
running 8-1, 8-10– 8-13
troubleshooting E-19
queries, inventory
CDROM_QUERY C-60
COMPUTER_MEM_QUERY C-61
FLPY_DRV_QUERY C-61
H_CDROM_QUERY C-79
H_COMPUTER_MEM_QUERY C-79
H_COMPUTER_QUERY C-79
H_FLPY_DRV_QUERY C-79
H_HDISK_QUERY C-79
H_HEADER_INFO_QUERY C-79
H_INST_FILE_QUERY C-79
H_INVENTORY_SWARE C-79
H_IP_ADDR_QUERY C-79
H_IPX_ADDR_QUERY C-79
H_MOUSE_QUERY C-79
H_NATIV_SWARE_QUERY C-79
H_NET_CARD_QUERY C-79
H_NOSIG_FILES_QUERY C-79
H_NW_SERVER_QUERY C-79
H_NW_VOLS_QUERY C-79
H_OS_QUERY C-79
H_PARTITION_QUERY C-79
H_PC_BIOS_QUERY C-79
H_PCI_DEV_QUERY C-79
H_PCPROCESSOR_QUERY C-79
H_PRINTER_QUERY C-79
H_PROCESSOR_QUERY C-79
H_STORAGE_DEV_QUERY C-79
H_TAPEDRV_QUERY C-79
H_UNIX_SYS_QUERY C-79
H_USB_DEV_QUERY C-80
H_VID_CARD_QUERY C-80
HDISK_QUERY C-62
HEADER_INFO_QUERY C-62
INST_FILE_QUERY C-63
INVENTORY_HWARE C-63
INVENTORY_SWARE C-64
IP_ADDR_QUERY C-65
IPX_ADDR_QUERY C-65
KEYBOARD_QUERY C-66
MATCH_CRC32_QUERY C-66
MATCH_MD5_QUERY C-67
MATCH_QUICK_QUERY C-67
MOUSE_QUERY C-68
NATIV_SWARE_QUERY C-68
NET_CARD_QUERY C-69
NOSIG_FILES_QUERY C-69
NW_SERVER_QUERY C-69
NW_VOLS_QUERY C-71
OS_QUERY C-71
PARTITION_QUERY C-72
PC_BIOS_QUERY C-72
PC_PROCESSOR_QUERY C-73

PCI_DEV_QUERY C-74
PRINTER_QUERY C-75
PROCESSOR_QUERY C-75
STORAGE_DEV_QUERY C-76
TAPEDRV_QUERY C-76
UNIX_SYS_QUERY C-77
USB_DEV_QUERY C-77
VID_CARD_QUERY C-78
queries, subscription
128MB_SUBSCRIPTIONS C-80
256MB_SUBSCRIPTIONS C-80
512MB_SUBSCRIPTIONS C-80
AIX_SUBSCRIPTIONS C-80
H_AIX_SUBSCRIPTIONS C-82
H_HPUX_SUBSCRIPTIONS C-82
H_NETWARE_SUBSCRIPTIONS C-82
H_OS2_SUBSCRIPTIONS C-82
H_Solaris_SUBSCRIPTIONS C-82
H_WIN_2000_SUBSCRIPTIONS C-82
H_WIN_98_SUBSCRIPTIONS C-82
H_WIN_ALL_SUBSCRIPTIONS C-82
H_WIN_ME_SUBSCRIPTIONS C-82
H_WIN_NT_SUBSCRIPTIONS C-82
HPUX_SUBSCRIPTIONS C-80
NETWARE_SUBSCRIPTIONS C-80
OS2_SUBSCRIPTIONS C-80
Solaris_SUBSCRIPTIONS C-81
SUB_128MB_SUBSCRIPTIONS C-80
WIN_2000_SUBSCRIPTIONS C-81
WIN_98_SUBSCRIPTIONS C-81
WIN_ALL_SUBSCRIPTIONS C-81
WIN_ME_SUBSCRIPTIONS C-81
WIN_NT_SUBSCRIPTIONS C-81
query libraries, creating 8-4
Query_edit role A-1
Query_execute role A-1
Query_view role A-1
QUERY_VIEWS table 7-35, 7-36
queue
completed 1-4
deferred 1-4, E-9
description 1-4
error 1-4, E-9
input 1-4, E-9
log files E-9
output 1-4, E-9
viewing information about 4-19, B-22

R

RDBMS
configuration 2-11
server 2-11
RDBMS Interface Module (RIM). *See* RIM host and RIM object
RECORD_ACTION, description C-58
RECORD_TIME, description C-6
region number 8-20
registered name, definition B-3
related documents xv
relational database management system. *See* RDBMS
relative path B-4
release notes xv
renaming profiles 5-29
repeater hierarchy 1-3, 2-4, 4-4
repeater site 1-3
resetting a collector B-13
resetting collection routes 2-4, 4-5, B-12
resource type, InventoryConfig 2-27, 5-6
restore role A-2
Results page 7-26, 7-29
retry 4-7, B-11

RIM host
choosing 2-11, 2-23
definition 1-3
interconnected TMRs 2-12
See also RIM object
with DB2 2-12
with Informix 2-15
with MS SQL Server 2-15
with Oracle 2-16
with Sybase 2-16
RIM object
changing password 2-28
configuring 4-11
creating 4-11
definition 1-5
inv_40 2-28
inv_query 2-28
invdh_1 2-28– 2-29, 4-11
inventory 2-28
See also RIM host
specifying application label 4-12
specifying maximum database connections 4-12
rim_db_log E-12
RIM_update role 7-23, A-1
RIM_view role 7-23, A-1
roaming endpoints. *See* mobile endpoints
roles. *See* authorization roles
Run Query dialog box 8-12
run-time directory
default location 4-5, B-10
moving 4-5, B-10

S

sa_config.log B-32, E-5
sa_results.log B-32, E-5
scalable collection 1-6
Scalable Collection Service. *See* SCS
scan notification
See status information
scanner
DMI 5-19
Tivoli hardware 5-19
scanning procedure 6-3– 6-8
scans
choosing hardware components 5-19, 5-23, B-86, B-100
collecting custom MIF files 5-23, B-81, B-95
customizing hardware 5-18– 5-23, B-86, B-100
customizing software 5-10– 5-18, B-90, B-104
endpoint-initiated 6-8, B-31
for .exe files only 5-17, B-90, B-104
for basic information 5-16, B-90, B-104
for BIOS information 7-38
header 5-16, B-90
hiding distributions to roaming endpoints B-28, B-78
initial scans 6-1
mandatory distributions to roaming endpoints B-27
PC hardware 5-18, B-86
reminder messages to roaming endpoints B-27
replace with current results 5-10, B-39
sending data to the configuration repository 5-12, B-90, B-104
setting data options 5-9, B-79
setting distribution deadline 6-5, B-26
setting distribution options 5-9, B-77
setting notification interval 6-5, B-26
setting priority 6-3, 6-4, B-25

setting timeout for endpoint response 6-6, B-26
setting timeout for profile distribution 6-5, B-26
setting timeout for scan attempt B-77
software signature 5-15, B-90, B-104
specifying custom scripts 5-23, B-81, B-95
specifying directories 5-17, B-81, B-95
specifying files 5-17, B-81, B-95
support for roaming endpoints B-29
troubleshooting E-13
UNIX hardware 5-22, B-100
update with differences 5-9, 6-2, B-39
viewing status 4-20, B-61
wake-on-LAN support B-29, B-79
scheduler daemon 1-4
scheduling collections 4-14, B-12
scripts
custom 5-23– 5-24, B-81, B-95
dmi_*RDBMS_type* _schema.sql 5-21, E-4
h_inv_db2_schema.sql 3-5
h_inv_infx_schema.sql 3-8
h_inv_ms_schema.sql 3-12
h_inv_ora_schema.sql 3-16
h_inv_syb_schema.sql 3-20
h_inventory_query.sh 3-22
h_subscription_query.sh 3-22
inv_db2_admin.sql 3-3
inv_db2_schema.sql 3-4
inv_infx_schema.sql 3-8
inv_ms_sql_admin.sql script 3-10
inv_ms_sql_schema.sql 3-11
inv_ora_admin.sql 3-13
inv_ora_schema.sql 3-15
inv_syb_admin.sql 3-17
inv_syb_schema.sql 3-19
inventory_query.sh 3-22
policy D-22
subscription_query.sh 3-22
Scripts and MIF Files window 5-23
SCS
commands B-6
components needed 1-3
controlling data flow 4-17
features 1-13
how it works with Tivoli Inventory 1-6, 4-2
installing from the command line 2-10
installing from the desktop 2-6
obtaining information about 4-18
troubleshooting E-8– E-19
security. *See* authorization roles
senior role A-2
SER_NUM (CD-ROM), description C-6
SER_NUM (hard drive), description C-11
SER_NUM (PC processor), description C-43
SER_NUM (processor), description C-47
SER_NUM (storage device), description C-48
SER_NUM (USB), description C-55
SER_NUM_ENABLED, description C-43
Set Login Names dialog box 7-10
Set TMR Roles dialog box 7-10
Signatures window 7-4
SIMD_EXT_SUPP, description C-43
size of data units 4-6, 4-17, B-10
Software Installation Service (SIS) 2-3
Software Scan Configuration dialog box 5-14
software signatures
creating 7-2– 7-5, B-67
definition 7-2
deleting 7-2– 7-5, B-67
editing 7-2– 7-5, B-67
installing 3-21
scanning with 5-15, B-90, B-104

Software window 5-10, 5-12
Solaris_SUBSCRIPTIONS C-81
starting a collector 4-5, B-9
status collector
configuring options 4-8, B-18, B-73
definition 1-4
how it works 1-10
viewing options 4-22, B-38– B-39
status information
configuring 4-8, 4-21, B-19, B-73
definition 4-8
scan notification 4-9
specifying host B-77
specifying log file B-77
specifying when it is sent B-78
specifying where it is sent B-77
viewing collector status 4-19, B-22
viewing scan status B-34
stopping a collector 4-5, B-9
STORAGE_DEV table C-100
STORAGE_DEV_QUERY C-76
STORAGE_DEV_VIEW C-48
STORAGE_TYPE (CD-ROM), description C-6
STORAGE_TYPE (floppy drive), description C-10
STORAGE_TYPE (hard drive), description C-11
STORAGE_TYPE (storage device), description C-48
SUB_128MB_SUBSCRIPTIONS C-80
subscribers
defining with queries 8-14
definition 1-5
subscription_query.sh script 3-22
super role A-2
SWARE_DESC (CRC32 match), description C-49
SWARE_DESC (MD5 match), description C-51
SWARE_DESC (Quick match), description C-53
SWARE_DESC (software match), description C-23
SWARE_DESC, description C-16
SWARE_MATCH_CRC32 C-49
SWARE_MATCH_MD5 C-51
SWARE_MATCH_QUICK C-53
SWARE_NAME (CRC32 match), description C-49
SWARE_NAME (MD5 match), description C-51
SWARE_NAME (Quick match), description C-53
SWARE_NAME (software match), description C-23
SWARE_NAME, description C-16
SWARE_SIG table C-100
SWARE_SIZE (CRC32 match), description C-50
SWARE_SIZE (MD5 match), description C-51
SWARE_SIZE (Quick match), description C-53
SWARE_SIZE (software match), description C-23
SWARE_SIZE, description C-16
SWARE_VERS (CRC32 match), description C-49
SWARE_VERS (MD5 match), description C-51
SWARE_VERS (Quick match), description C-53
SWARE_VERS (software match), description C-23
SWARE_VERS, description C-16
SWSIGS.INI 3-21

Sybase
configuration 2-16
creating a configuration repository 3-17– 3-20
RIM host 2-16
syntax, for commands B-2

T

tables
adding to the configuration repository 7-20
cascading delete 7-21
default C-83– C-105
designing custom 7-14– 7-15
for DB2 and Informix RDBMSs 7-14, 7-16
history. *See* history tables
naming 7-14
operational data C-2
tables, configuration repository
COMPUTER C-83
COMPUTER_SYS_MEM C-84
FILE_DESC C-85
FILE_FILTER C-85
FILE_PATH C-86
H_COMPUTER C-104
H_COMPUTER_SYS_MEM C-104
H_INST_HEADER_INFO C-104
H_INST_MOUSE C-104
H_INST_NATIVE_SWARE C-104
H_INST_PARTITION C-104
H_INST_PRINTER C-104
H_INST_PROCESSOR C-104
H_INST_USB_DEV C-104
H_INST_VID_CARD C-104
H_IP_ADDR C-104
H_IPX_ADDR C-104
H_MATCHED_SWARE C-104
H_NET_ADAPTER C-104
H_NW_SERVER C-104
H_NW_VOLS C-104
H_PC_SYS_PARAMS C-104
H_PCI_DEV C-104
H_STORAGE_DEV C-104
H_UNIX_SYS_PARAMS C-105
H_UNMATCHED_FILES C-105
HDISK C-86
HEADER_INFO C-87
INST_HEADER_INFO C-87
INST_MOUSE C-88
INST_NATIV_SWARE C-88
INST_PARTITION C-89
INST_PRINTER C-89
INST_PROCESSOR C-90
INST_USB_DEV C-90
INST_VID_CARD C-91
IP_ADDR C-92
IPX_ADDR C-92
MATCHED_SWARE C-93
MOUSE C-93
NATIV_SWARE C-94
NET_ADAPTER C-94
NW_SERVER C-95
PC_SYS_PARAMS C-96
PCI_DEV C-97
PRINTER C-98
PROCESSOR C-98
STORAGE_DEV C-100
SWARE_SIG C-100
UNIX_SYS_PARAMS C-101
UNMATCHED_FILES C-102
USB_DEV C-103
VID_CARD C-103
tail E-10, E-11, E-12
TAPEDRV_QUERY C-76
targets. *See* subscribers
tasks performed by Tivoli Inventory 1-2
TEC
sending status information 4-8, 4-22, B-38, B-78
using with Tivoli Inventory 2-35

testing configurations 3-6, 3-9, 3-12, 3-16, 3-20
text conventions xviii
threads
configuring for collectors 4-7, 4-18, B-11
configuring for inventory data handler 4-9
TIME_STAMP_COUNTER, description C-41
Timeout Settings dialog box 6-5
Tivoli desktop 2-6
Tivoli Enterprise Console. *See* TEC
Tivoli hardware scanner 5-19
Tivoli Hardware Scanner dialog box 5-19, 5-23
Tivoli Inventory
coexistence with previous versions 2-2
commands B-4
components 1-2
features 1-10
how it works 1-5
how it works with SCS 1-6
notice group 6-2
security 1-14
tasks performed 1-2
uninstalling 2-33
UserLink 1-3
Web-based interface 1-3
Tivoli Inventory Gateway, installing 2-31
Tivoli Management Framework, description 1-1
Tivoli scheduler 4-15, 4-17
TME 10 Web Access page 7-23
TME_OBJECT_ID, description C-6
TME_OBJECT_LABEL, description C-6
tmersrvd account 4-6, B-10
TMR
considerations for interconnected TMRs 4-22
number of inventory data handlers per B-18
queries in interconnected TMRs 3-23
resources of the same name and type within a TMR B-3
RIM host for interconnected TMRs 2-12
tnsnames.ora file 2-16, 2-26
TOTAL_PAGES, description C-7
transmission chunk 4-6, 4-17, B-10
troubleshooting
configuration repository E-1
DMI E-1– E-4
endpoint-initiated scans E-4– E-6
GUI E-6– E-8
queries E-19
SCS E-8– E-19
typefaces, use of xviii

U

uninstalling Tivoli Inventory 2-33
UNIX hardware scans 5-22, B-100
UNIX_SYS_PARAMS table C-101
UNIX_SYS_QUERY C-77
UNMATCHED_FILES table C-102
USB_DEV table C-103
USB_DEV_QUERY C-77
USB_DEV_VIEW C-55
USB_VERS, description C-55
User Data Form 7-30, 7-33
User Data Template
adding MIF attributes 7-24– 7-30
adding MIF groups 7-24– 7-30
page 7-24

user information
gathering 7-6– 7-34
retrieving and saving 7-34
viewing 7-34
user role A-1
USER_NAME, description C-38
useradd.mif file
creating 7-6– 7-34
viewing groups and attributes 7-30
UserLink for Tivoli Inventory page 7-33
UserLink for Tivoli page 7-32
UserLink, enabling access 7-8– 7-13
userlink.htm file 7-31

V

VENDOR_ID, description C-56
VID_BIOS_RELDATE, description C-57
VID_CARD table C-103
VID_CARD_BIOS, description C-57
VID_CARD_MODEL, description C-57
VID_CARD_QUERY C-78
VID_CARD_VIEW C-57
VID_CHIP_TYPE, description C-57
VID_COLORS, description C-58
VID_DAC_TYPE, description C-57
VID_HORIZNTL_RES, description C-58
VID_MEM, description C-57
VID_VERTICAL_RES, description C-58
viewing inventory information
about one client 8-16
procedure 8-16– 8-20
wqueryinv B-71
views
adding 7-34
CDROM_VIEW C-6
COMPUTER_MEM_VIEW C-7
COMPUTER_VIEW C-9
custom 7-34
default, list of C-5– C-60
FLPY_DRV_VIEW C-10
H_CDROM_VIEW C-59
H_COMPUTER_VIEW C-59
H_FLPY_DRV_VIEW C-59
H_HDISK_VIEW C-59
H_HEADER_VIEW C-59
H_INST_FILE_VIEW C-59
H_INST_SWARE_VIEW C-59
H_IP_ADDR_VIEW C-59
H_IPX_ADDR_VIEW C-59
H_MEM_VIEW C-59
H_MOUSE_VIEW C-59
H_NATIV_VIEW C-59
H_NET_CARD_VIEW C-59
H_NOSIG_FILES_VIEW C-59
H_NW_SERVER_VIEW C-59
H_NW_VOLS_VIEW C-59
H_OS_VIEW C-59
H_PARTITION_VIEW C-59
H_PC_BIOS_VIEW C-59
H_PCI_DEV_VIEW C-59
H_PCPROCESSOR_VIEW C-59
H_PRINTER_VIEW C-59
H_PROCESSOR_VIEW C-59
H_STORAGE_DEV_VIEW C-59
H_TAPEDRV_VIEW C-60
H_UNIX_SYS_VIEW C-60
H_USB_DEV_VIEW C-60
H_VID_CARD_VIEW C-60
HDISK_VIEW C-11
HEADER_INFO_VIEW C-13
historical C-58– C-60
INST_FILE_VIEW C-14
INST_SWARE_VIEW C-16
INVENTORYDATA C-17
IP_ADDR_VIEW C-19
IPX_ADDR_VIEW C-21
KEYBOARD_VIEW C-22
MATCH_SWARE_VIEW C-23
MOUSE_VIEW C-24
NATIV_SWARE_VIEW C-25

views, *continued*
NET_CARD_VIEW C-26
NOSIG_FILES_VIEW C-28
NW_SERVER_VIEW C-29
NW_VOLS_VIEW C-32
OS_VIEW C-34
PARTITION_VIEW C-36
PC_BIOS_VIEW C-38
PC_PROCESSOR_VIEW C-40
PCI_DEV_VIEW C-44
PRINTER_VIEW C-45
PROCESSOR_VIEW C-47
STORAGE_DEV_VIEW C-48
SWARE_MATCH_CRC32 C-49
SWARE_MATCH_MD5 C-51
SWARE_MATCH_QUICK C-53
USB_DEV_VIEW C-55
VID_CARD_VIEW C-57
VIRT_FREE_KB, description C-8
VIRT_MODE_EXT, description C-41
VIRT_TOTAL_KB, description C-7

W

wadminep E-2, E-14
wake-on-LAN
description 1-12
enabling B-29, B-79
WAN connections 2-5, 4-22
wbkupdb 2-17
wcollect 2-4, 4-4– 4-9, 4-16– 4-19, B-8, E-10– E-19
wcomprules 2-36
wcprb 2-36
wcrtadmin 7-12
wcrtinvcb 4-11, B-16
wcrtinvdh 4-8, 4-10, 4-22, B-18, B-34– B-35, B-73
wcrtpol D-3
wcrtprf 5-7, 5-26, 5-27
wcrtquery 7-22
wcrtrb 2-36
wcrtrim 2-28, 4-12
wcstat 4-19, B-13, B-22, E-9
wdel 4-10, 4-11, 5-29
wdistinv 6-7, B-25
Web pages, accessing 7-24
wepscan 6-8, B-31, E-5
wgetinvdh 4-20, B-34, E-15
wgetinvglobal 4-21, 4-22, B-37
wgetinvpcfiles B-41
wgetinvpchw B-43
wgetinvpcsw B-46
wgetinvunixfiles B-51
wgetinvunixhw B-53
wgetinvunixsw B-56
wgetpolm D-4
wgetrim 2-12, E-15
wgetscanstat 4-20, 4-22, B-61, E-13– E-14
wimprbclass 2-36
WIN_2000_SUBSCRIPTIONS C-81
WIN_98_SUBSCRIPTIONS C-81
WIN_ALL_SUBSCRIPTIONS C-81
WIN_ME_SUBSCRIPTIONS C-81
WIN_NT_SUBSCRIPTIONS C-81
window
Hardware 5-18, 5-22
Profile Manager 5-4, 5-7
Scripts and MIF Files 5-23
Signatures 7-4
Software 5-10, 5-12
winstall 2-29
winvfilter B-63
winvrmnode 2-35, 7-21, B-65
winvsig 3-21, 7-3, 7-4, B-67
wloadrb 2-36
wlookup 4-13

wlsinst E-15
wlspolm D-4
wmdist 6-6
WMI scanner 1-11
WORKGROUP_NAME, description C-38
wpatch 2-10
wputpolm D-5
wqueryinv B-71
wrimset E-20
wrimtest E-14, E-19
wrimtrace E-12
wrpt 4-4, 4-5, B-12, E-15, E-16
wruninvquery 8-16
wrunquery 8-11, 8-13, 8-16
wsetadmin 7-13
wsetinvdh 4-8, 4-21– 4-22, B-19, B-34– B-35, B-73
wsetinvglobal 4-21– 4-22, B-18, B-19, B-21, B-35, B-38, B-73– B-74, B-76, E-11
wsetinvpcfiles B-81
wsetinvpchw B-86
wsetinvpcsw B-90
wsetinvunixfiles B-95
wsetinvunixhw B-100
wsetinvunixsw B-104
wsetpr D-6
wsetrim 2-12, 4-14
wsetrimpw 2-12, 2-29, 3-4, 3-8, 3-19
wstartesvr 2-37
wstopesvr 2-36
wuninst 2-33